

# Trabajo Fin de Grado Grado en Ingeniería de las Tecnologías de Telecomunicación

## Asistente de voz para la cobertura de eventos electorales municipales

Autor: Sergio Olivares Rodríguez

Tutor: Rubén Martín Clemente

**Dpto. Teoría de la Señal y Comunicaciones  
Escuela Técnica Superior de Ingeniería  
Universidad de Sevilla**

Sevilla, 2021





Trabajo Fin de Grado  
Grado en Ingeniería de las Tecnologías de Telecomunicación

# **Asistente de voz para la cobertura de eventos electorales municipales**

Autor:  
Sergio Olivares Rodríguez

Tutor:  
Rubén Martín Clemente  
Profesor Titular

Dpto. Teoría de la Señal y Comunicaciones  
Escuela Técnica Superior de Ingeniería  
Universidad de Sevilla

Sevilla, 2021



Trabajo Fin de Grado: Asistente de voz para la cobertura de eventos electorales municipales

Autor: Sergio Olivares Rodríguez  
Tutor: Rubén Martín Clemente

El tribunal nombrado para juzgar el trabajo arriba indicado, compuesto por los siguientes profesores:

Presidente:

Vocal/es:

Secretario:

acuerdan otorgarle la calificación de:

El Secretario del Tribunal

Fecha:



*A los locos*  
*A los inadaptados*  
*A los rebeldes*  
*A los problemáticos*  
***Los que ven las cosas de manera diferente***





# Agradecimientos

---

**M**e gustaría comenzar estos agradecimientos acordándome de por qué estoy aquí. Creo que es importante señalar que por el país en el que me ha tocado nacer, el siglo en el que me ha tocado desarrollarme y la familia en la que he crecido soy un absoluto privilegiado. Y cuando las cosas se ponen difíciles a veces destacar esto lo relativiza todo.

Hay muchas personas que me han podido acompañar a lo largo de estos cinco años en la ETSI: Jose, Pablo, Jesús, Fernando, Daniel... pero me quiero acordar de una en especial, y esa es Ana. Ha sido compañera real de batallas en la universidad pero también en Sevilla. No sé qué habría sido de mí todos estos años sin ella.

Al Erasmus que decidí coger hace ya dos años, que se convirtió en la mejor experiencia de mi vida y que me enseñó que la vida era algo más, y que yo podía ser alguien más. A Aurora que en esos meses se convirtió en compañera indispensable en Bay Campus.

A los profesores que durante mi etapa en el colegio y en el instituto me valoraron y me quisieron en mi diferencia. Son ellos los que han conseguido que siguiera creyendo en mí aunque en algunos momentos de mi etapa universitaria sintiera que no era lo suficientemente bueno. Mis sinceros agradecimientos a Rubén, el tutor de este trabajo, por su disposición plena, su confianza en mí y por darme la oportunidad e idea para hacer el proyecto.

A mi familia: mis padres y mis abuelos. Por el amor sin más, amor porque sí. Y en especial a mi madre por dar su vida y confiar siempre en mí.

Gracias por supuesto a mi hermano Pablo. Él ha sido apoyo incansable, la persona que más cree en mí del universo y a la que nunca le podré agradecer tanto la paciencia que siempre ha tenido conmigo. Él es brillantez, ingenio y astucia; pero también predisposición y bondad. Esta universidad se queda con un diamante en bruto, ojalá lo sepan valorar.

Y por último, gracias a mi novio, Álvaro, por ser compañero de vida, de alma, de ideas y de corazón.

A todos, gracias.

*Sergio Olivares Rodríguez  
Madrid, 2021*



# Resumen

---

Este proyecto trata de aplicar las técnicas de Inteligencia Artificial existentes para la generación automática de texto en el desarrollo de una aplicación de asistente de voz, cuyo objetivo será el dar cobertura informativa en elecciones municipales. La aplicación sería capaz de brindar información automática para cualquiera de los más de 8000 municipios que componen España, además de responder a preguntas del usuario sobre estas cuestiones.

Se han utilizado los modelos de lenguaje GPT-2, T5 y BERT para tareas de generación de textos, resumen de textos y resolución de preguntas durante todo el desarrollo del trabajo, así como múltiples plataformas online como DialogFlow o HuggingFace. Todo el código ha sido escrito en Python.

La memoria incluye tanto introducciones teóricas sobre los ámbitos de aplicación como explicaciones y descripciones detalladas de los procedimientos llevados a cabo para la implementación de la aplicación, junto con capturas del resultado final en funcionamiento.



# Abstract

---

This project aims to apply existing Artificial Intelligence techniques for Natural-language Generation in the development of a voice assistant app, whose main goal is to provide media coverage in local elections. The application would be able to inform of any of the more than 8000 towns in Spain, as well as answer user's questions about these matters.

The language models GPT-2, T5 and BERT have been used for text generation tasks, text summarization and question answering throughout the development of the project, besides many online platforms such as DialogFlow or HuggingFace. All the code has been written in Python.

This report includes theoretical introduction to the application field, but also detailed explanations and descriptions about the methods carried out for the fulfillment of the app. It will also include different screenshots of it.



# Índice

---

<i>Resumen</i>	V
<i>Abstract</i>	VII
<b>1 Introducción</b>	<b>1</b>
1.1 Objetivos	1
1.2 Estructura de la memoria	2
<b>2 Inteligencia Artificial Aplicada a la Generación de Textos</b>	<b>3</b>
2.1 Procesamiento de Lenguaje Natural	3
2.2 Redes Neuronales Artificiales	4
2.3 Transferencia de Aprendizaje	5
<b>3 Modelos de Lenguaje Utilizados</b>	<b>7</b>
3.1 Arquitectura <i>Transformer</i>	7
3.2 <i>Generative Pre-trained Transformer 2</i> (GPT-2)	9
3.3 <i>Text-To-Text Transfer Transformer</i> (T5)	9
3.4 <i>Bidirectional Encoder Representations from Transformers</i> (BERT)	10
<b>4 Aplicación en la Cobertura de Eventos Electorales Municipales</b>	<b>11</b>
4.1 Motivación y Antecedentes	11
4.1.1 Generación automática de textos periodísticos	11
4.1.2 Eventos Electorales Municipales	11
4.1.3 Asistentes Virtuales	12
4.2 Funcionalidad	13
4.3 Componentes	14
4.3.1 Bibliotecas y frameworks utilizados	14
4.4 Proceso de diseño e implementación	16
4.4.1 Generación de Texto	17
Acceso a los modelos	21
4.4.2 Obtención y Procesado de Datos	21
4.4.3 Interacción del Usuario con el Asistente Virtual	22
Gestión del tiempo de elaboración de los resultados	24
<b>5 Ejemplos de validación</b>	<b>27</b>
5.1 Datos de ejemplo utilizados	27
5.2 Caso 1: 2 partidos de los cuales 2 estatales	27
5.3 Caso 2: 5 partidos de los cuales 3 estatales, 1 regional, 1 local	30
5.4 Caso 3: 8 partidos de los cuales 6 estatales, 2 regionales	32
<b>6 Conclusiones y Desarrollos Futuros</b>	<b>35</b>

<b>Apéndice A CÓDIGOS</b>	<b>37</b>
<i>Índice de Figuras</i>	55
<i>Índice de Tablas</i>	57
<i>Bibliografía</i>	59



# 1 Introducción

---

La tecnología es exitosa cuando consigue ponerse de tú a tú con las personas y formar parte de su día a día como un elemento más. Cuando consigue fundirse con aquellas cosas que amamos en la vida y las hace más sencillas, más inteligentes, más conectadas. Para mí, es en esta intersección donde creo que lo que los ingenieros hacemos tiene sentido y cambia vidas. Con esta idea nace este proyecto, con mi voluntad de no hacer un TFG cualquiera y darle una vuelta para ver qué podía aportar con este humilde trabajo.

Esta aplicación podrá informar de los resultados de las elecciones municipales para todos y cada uno de los más de 8000 municipios de este país, en tiempo real y de una forma tan accesible como a un golpe de voz, desde un asistente de voz. Pero, ¿y qué utilidad tiene esto?

Vivimos en el momento de la historia en el que más conectados, informados y cultos somos. Sin embargo, las conocidas como "fake news", noticias falsas, no dejan de propagarse de forma rápida por nuestros canales de información y acaban sirviendo en muchos casos para manipular a la sociedad por parte de medios, empresas y partidos políticos interesados. Por tanto, es indispensable que se sigan creando herramientas para que la información veraz sea más accesible y pueda contrarrestar a estas otras fuentes falsas.

Para ello nos ayuda la Inteligencia Artificial, para llegar a aquellos lugares donde los humanos no podemos. La aplicación que se presenta en este proyecto no tiene la idea de sustituir a ningún periodista, porque su trabajo es imposible de hacer por un humano. Tiene el fin de ser un aliado en el periodismo y que mientras la máquina se encarga de estas tareas tediosas, ellos puedan usar su talento para aquellos lugares donde la tecnología nunca podrá llegar: la creación de arte.

Además, el uso de un asistente de voz creo que resulta de gran utilidad. Estos dispositivos cada vez están más extendidos, y seguirán haciéndolo. No tienen complicadas interfaces de usuario, ni pantallas donde leer información. Simplemente nos comunicamos a ellos como lo haríamos con otras personas. De esta forma, son más rápidos, sencillos y accesibles. Podremos preguntar por los datos de las elecciones estemos duchándonos, conduciendo o desayunando; seamos niños, adultos o ancianos; sin ni siquiera tener un teléfono móvil a mano, si disponemos de un altavoz inteligente.

A lo largo del proyecto, se podrá ver el alucinante rendimiento que tienen los modelos de lenguaje para la generación automática de texto, el corazón de la aplicación. Es gracias a ellos el éxito en funcionamiento de este proyecto, y es un hito en la tecnología. Para mí ha sido un tremendo placer poder adentrarme en estas técnicas durante el desarrollo del trabajo, y deseo que durante mi trayectoria laboral pueda seguir conociendo más de este ámbito, el cual me parece prometedor.

## 1.1 Objetivos

Este proyecto tiene en su base los siguientes objetivos:

- El principal, el más importante, el desarrollo de una aplicación para generación de textos aplicada a la producción de contenidos informativos que sean destinados a la cobertura de elecciones municipales. Esto incluye la búsqueda de los mejores modelos para ello, su entrenamiento para nuestra tarea y su aplicación en una plataforma que fuera útil para su uso.
- El segundo es que a lo largo de todo este período con el proyecto, se pueda aprender de los campos de Inteligencia Artificial, el Procesamiento de Lenguajes Naturales y la Generación Automática de Texto. Para mí, recién salido del grado, estos conocimientos me serán seguro de gran ayuda en mi carrera laboral.

- El tercero será la elaboración de una memoria explicativa, esta misma, donde se expliquen los contenidos teóricos necesarios para el desarrollo del trabajo así como los pasos que se han seguido para la implementación de la aplicación. También se deben mostrar los resultados finales y las conclusiones a las que se ha llegado
- Por último, es también objetivo de este proyecto la divulgación del mismo para una posible mejora en su futuro, pues se trata de un proyecto preliminar y en fase de pruebas, que tendría aún mucho camino que recorrer.

## **1.2 Estructura de la memoria**

Este trabajo escrito se estructura en capítulos, cada uno de los cuales tiene su función y da sentido al proyecto completo. El texto explicativo será intercalado con figuras visuales de autoría propia y capturas de las distintas herramientas utilizadas y de la propia aplicación.

En el **capítulo 2** se hace una introducción teórica del campo de la Inteligencia Artificial, enfocándonos en aquellas técnicas que vamos a usar, tales como las redes neuronales o la Transferencia de Aprendizaje.

A continuación, en el **capítulo 3**, hay una continuación teórica centrándose en los modelos de lenguaje utilizados durante el desarrollo del proyecto y la arquitectura en la que se basan todos.

En uno de los capítulos más importantes, el **capítulo 4**, se hará un desarrollo explicativo de la aplicación objeto de este trabajo. Se comienza con alguna otra introducción teórica de la motivación del proyecto en el ámbito humano aplicado, el de la cobertura de las elecciones municipales, para seguir comentando los componentes de la aplicación y terminar explicando los mismos junto con una documentación de los pasos seguidos.

El **capítulo 5** se trata de un desarrollo de los resultados, con tres casos de ejemplos de validación junto con las capturas de la aplicación, en la que se muestran los éxitos y debilidades de la misma.

Y, finalmente, el **capítulo 6** describe las conclusiones a las que se ha podido llegar con la elaboración del proyecto y qué ampliaciones o mejoras quedarían pendientes en el mismo.

## 2 Inteligencia Artificial Aplicada a la Generación de Textos

---

En este capítulo se introducirá la base teórica que ha servido como apoyo para la realización de este proyecto. Todos los conceptos que se van a presentar pertenecen al campo de la Inteligencia Artificial, la cual tiene como objetivo, según definiría el matemático Alan Turing en 1950, diseñar máquinas capaces de exhibir un comportamiento inteligente; comparable o indistinguible del de un ser humano.

Durante años, había sido en ámbitos como la literatura o la poesía donde la tecnología había encontrado sus mayores limitaciones debido a su carácter lógico-matemático, muy alejado de la expresividad artística natural presente en este tipo de textos. No obstante, el desarrollo reciente de novedosas técnicas han permitido romper estas barreras: hemos conseguido que las máquinas puedan entender el lenguaje como nosotros (**Procesamiento de Lenguaje Natural**), que aprendan como nosotros (**Redes Neuronales Artificiales**) y que se transmitan los conocimientos de unas a otras como nosotros (**Transferencia de Aprendizaje**).

### 2.1 Procesamiento de Lenguaje Natural

En la introducción anterior se ha comentado cómo los ordenadores tienen formas de comunicarse muy diferentes a las de las personas. Ellos utilizan lo que se conoce como "lenguajes artificiales", que se caracterizan por estar gobernados por reglas gramaticales bien determinadas. Algunos ejemplos son las ecuaciones matemáticas o los lenguajes de programación. Por otro lado están los "lenguajes naturales", que son los que utilizamos las personas en nuestro día a día (español, inglés, francés, etc.). Se diferencian con los anteriores en que al tener un origen muy antiguo y haber evolucionado de generación en generación son más complicados de definir y, por tanto, las máquinas tienen más dificultades en entenderlos.

Así, el Procesamiento de Lenguaje Natural (PLN) es una rama de la Inteligencia Artificial que trata de transformar el lenguaje natural en un lenguaje artificial inteligible para un ordenador, para poder después realizar aplicaciones a partir de este entendimiento. Para ello, en primer lugar los sistemas que implementen estas tecnologías deberán ser capaces de analizar textos de entrada, reconocer los vocablos y palabras que contengan e interpretar el significado de las oraciones dentro de su contexto. Tras ello, ya podrán tomar decisiones y responder a estas entradas. Estas respuestas sufrirán de nuevo un procesamiento similar al de entrada, pero en el sentido contrario [1]. En la Figura 2.1 se muestra una posible arquitectura de este tipo de sistemas.

Este campo tiene aplicación en múltiples sectores y, como se ha expuesto, está alumbrando los mayores avances de la tecnología en los últimos años. Concretamente, el PLN es muy útil en el procesamiento del Big Data, es decir, toda aquella masa enorme de datos no estructurados que las organizaciones manejan hoy en día. Algunas de ellas son [2]:

- Búsqueda avanzada de información
- Reconocimiento de entidades nombradas (*NER*)
- Detección de temas o patrones en los textos
- Asistentes virtuales
- Análisis de sentimiento
- etc.

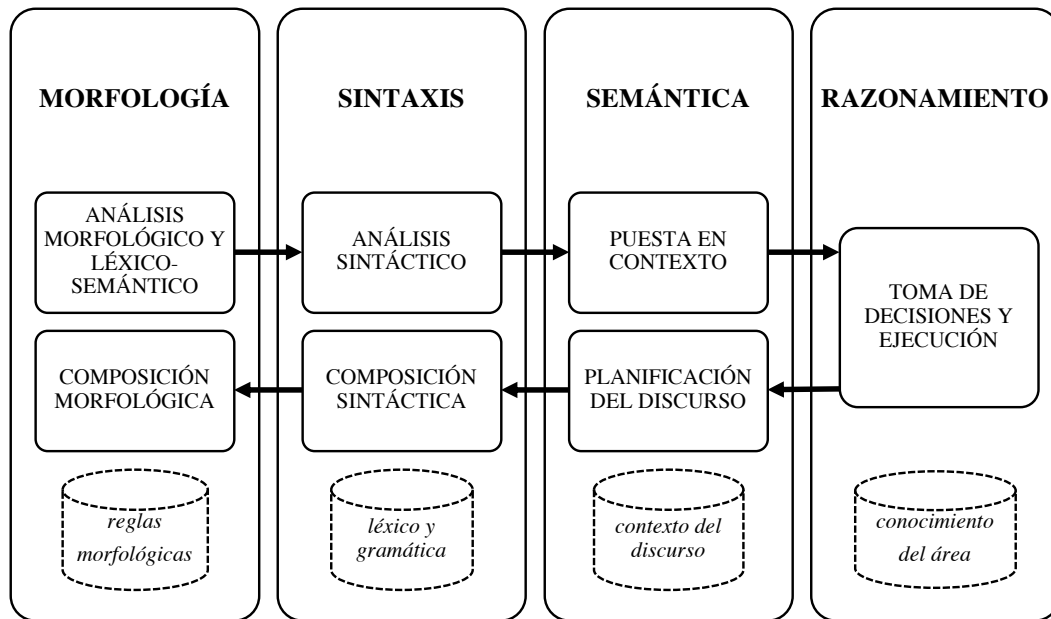


Figura 2.1 Arquitectura simplificada de un sistema de PLN.

## 2.2 Redes Neuronales Artificiales

Las redes neuronales son modelos computacionales basados en el funcionamiento del sistema nervioso humano. Su objetivo es aprender modificándose automáticamente a sí mismas de forma que pueda realizar tareas complejas que no podrían ser realizadas mediante programación clásica. Por tanto, tienen grandes aplicaciones en el campo que nos ocupa: el Procesamiento de Lenguaje Natural.

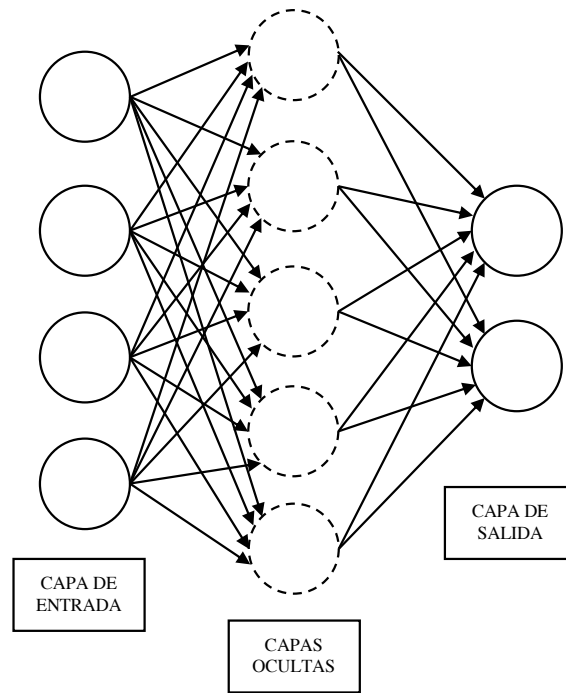
Consisten en un conjunto de unidades básicas, las neuronas, conectadas entre sí para transmitirse señales y organizadas en capas. Generalmente se pueden distinguir tres partes en una red neuronal [3]:

- Una capa de entrada (*Input Layer*)
- Una o varias capas ocultas (*Hidden Layer*)
- Una capa de salida (*Output Layer*)

Un esquema de esta estructura se puede ver en la Figura 2.2.

La idea de funcionamiento de estos modelos es que cada neurona está conectada con otras a través de enlaces en los que el valor de salida de la neurona se multiplica por un parámetro (peso). Además, puede también existir una función limitadora o umbral (función de activación) que modifica el valor resultado de salida o impone un límite que no debe sobrepasar [4]. Así, la información de entrada atraviesa la red neuronal, sometándose a estas operaciones, y produce unos valores de salida que serán la predicción calculada por la red.

No obstante, para conseguir que una red neuronal realice las funciones deseadas, es necesario entrenarla. Para ello, se presentan a la red ejemplos de datos de entrada para los que se conoce el resultado de salida. La red genera su propia predicción y la compara con los resultados conocidos. De esta forma, se modifican los pesos de las neuronas según el error obtenido y cuánto haya contribuido cada neurona a dicho resultado. Una vez entrenada, la red se podrá aplicar a casos futuros en los que se desconozca el resultado.



**Figura 2.2** Esquema de la estructura en capas de una red neuronal simplificada.

## 2.3 Transferencia de Aprendizaje

Las técnicas de Transferencia de Aprendizaje o *Transfer Learning* consisten en reutilizar el conocimiento adquirido en el entrenamiento de una red neuronal para un problema concreto y aplicarlo en un problema nuevo [5]. Por ejemplo, un modelo entrenado para reconocer animales podría ser el punto de partida para un modelo que necesitaríamos que clasificara imágenes de perros según su raza.

Así, la idea de estos métodos es poder usar todo el conocimiento que una red ha podido aprender para un problema para el que existe un amplísimo dataset de entrenamiento en una nueva tarea para la que no dispongamos de tanta información de prueba. Para este caso concreto el procedimiento sería el siguiente [6]:

1. Carga de la red pre-entrenada: contendrá unas capas iniciales que han aprendido a entender características básicas de nuestro contexto de aplicación junto con unas capas finales diseñadas para realizar la tarea específica para la que sí se disponía de un amplio dataset.
2. Reemplazo de las capas finales: por nuevas capas destinadas a aprender a realizar nuestro problema específico.
3. Entrenamiento de la red: con los datos y opciones diseñados para nuestra tarea final, los cuales como comentábamos eran limitados.
4. Predicción y evaluación de la precisión de la red: alimentando la red con datos de prueba, de forma que podamos ver cómo responde.
5. Despliegue de resultados: a partir de los resultados obtenidos en el punto anterior. Si no estuviésemos satisfechos con ellos, volveríamos al punto 3.

Un caso típico de PLN y Transferencia de Aprendizaje consiste en pre-entrenar un modelo con datos de texto sin etiquetar, los cuales están disponibles en abundancia, con una tarea que puede ser autosupervisada por la máquina (rellenar palabras que faltan, por ejemplo). Tras ello, la red neuronal puede ser ajustada con dataset de texto etiquetados más pequeños, lo cual resulta en un mayor rendimiento que si entrenáramos al modelo únicamente con el dataset pequeño.

Las ventajas de este tipo de técnicas se resumen en el ahorro de tiempo, la mejora del rendimiento de los modelos y la falta de necesidad de amplios dataset de entrenamiento. Así, desde que se empezaron a aplicar han supuesto un avance en el campo de la comprensión automática del lenguaje, pero también en otros como en la medicina. Por ejemplo, se ha usado para el descubrimiento de nuevos subtipos de cáncer [7], lo que ha permitido que se puedan orientar mejor los tratamientos de los pacientes.



## 3 Modelos de Lenguaje Utilizados

---

En este capítulo se introducirán los modelos de lenguaje utilizados en este proyecto para la generación de texto. Se basan en distribuciones estadísticas que predicen la validez de una secuencia de palabras concretas.

Las tres herramientas están desarrolladas por empresas privadas, pero se tratan de sistemas de código abierto, por lo que se pueden utilizar libremente para investigación, enseñanza, aplicaciones comerciales, etc. Esto ha traído algunos debates en la red, pues el hacer estos modelos públicos y disponibles para cualquier persona se temía por un posible mal uso, aplicándolo por ejemplo en la creación de "fake news" o spam [8].

Se apreciará durante este capítulo que los tres tienen muchos puntos en común, pero el principal es que todas se basan en la arquitectura conocida como *Transformer* que será también explicada a continuación.

### 3.1 Arquitectura *Transformer*

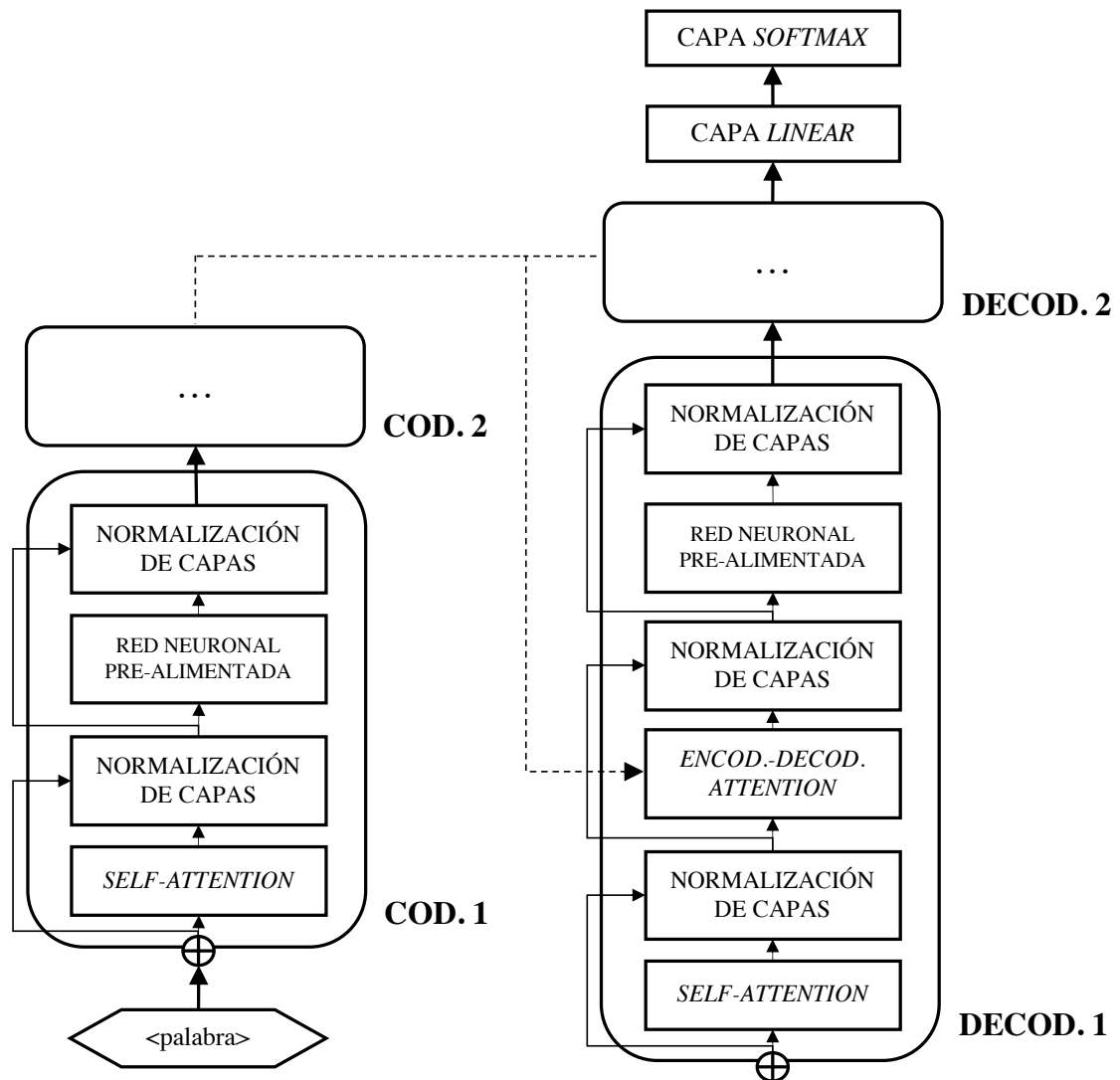
Este tipo de arquitectura se caracteriza por implementar el mecanismo de atención [9], evaluando la relevancia de cada parte de los datos de entrada respecto del conjunto.

En una primera instancia, se puede ver un modelo *Transformer* como una caja negra que tiene una entrada, por ejemplo una oración en un idioma, y una salida, que sería la traducción de esa oración en otro idioma. Partiendo de aquí, sus componentes serían [10]:

1. **Codificador (*encoder*):** que realmente será una pila de varios codificadores. Cuando entrenamos al modelo, por estos elementos se introducen los datos de entrada, es decir, que si tuviéramos un modelo para traducir del inglés al español aquí entrarían las palabras en español [11]. Cada uno de los codificadores reciben una lista de vectores que representan esas palabras y estos son procesados por los siguientes elementos:
  - a) Una capa de *Self-Attention*: ayuda al codificador a fijarse en el contexto completo de la oración de entrada cuando trata de entender la función de una palabra específica. Por ejemplo, al analizar la palabra "dormilona" de "mi madre se duerme al instante cuando se tumba en la cama, es muy dormilona", esta capa tendría la función de entender el contexto completo de la oración y detectar que "dormilona" se refiere a "mi madre" y no a "la cama".
  - b) Una Red Neuronal Pre-alimentada (*Feed Forward Neural Network*): que se encargará de procesar los vectores de texto que le lleguen, de forma que se ajusten mejor al siguiente codificador.
2. **Descodificador (*decoder*):** que será una pila del mismo número de descodificadores que de codificadores. A la hora del entrenamiento, por aquí entrarían los resultados correctos que el modelo comparará con los suyos para ir aprendiendo. En el caso del ejemplo del traductor, se introducirían las palabras correctamente traducidas en español. Cada uno contiene:
  - a) Una capa de *Self-Attention*: igual que en el codificador
  - b) Una capa de *Encoder-Decoder Attention*: aquí es donde se procesa la entrada de los codificadores (en el ejemplo, palabras en inglés), para aplicarles la relación que deberían tener con la salida de los descodificadores (palabras en español), la cual la ha aprendido tras fallar en iteraciones anteriores y modificar sus parámetros.

- c) Una Red Neuronal Pre-alimentada (*Feed Forward Neural Network*): de nuevo, procesará los vectores que le lleguen de forma que tengan una forma que se ajuste mejor al siguiente decodificador.

El último decodificador devolverá un vector, el cual será procesado por dos últimas capas para obtener el resultado final. La capa *Linear* es una última red neuronal que expandirá las dimensiones del vector hasta el número total del diccionario del modelo (en el ejemplo, el número total de palabras en español), en el que cada elemento representará la puntuación para una palabra del dataset. Es decir, cuánto más puntuación tenga una palabra, más probable será que sea la respuesta correcta según el criterio del modelo. Por otro lado, la capa *Softmax* transforma esas puntuaciones en probabilidades y la celda con la mayor probabilidad es la escogida para ser el resultado final. La Figura 3.1 muestra un diagrama de los componentes que forman este tipo de modelos.



**Figura 3.1** Diagrama secuencial de los componentes de un modelo de lenguaje basado en la arquitectura *Transformer* [10].

Ese resultado final es al que se le aplica la función de pérdidas, la cual, cada vez que ejecutemos una iteración del modelo, lo compara con la respuesta correcta que deberemos tener dentro de nuestro dataset de entrenamiento (entrada de los decodificadores). Entonces, modificaremos todos los parámetros del modelo utilizando la técnica de propagación hacia atrás.



### 3.2 Generative Pre-trained Transformer 2 (GPT-2)

Creado por la compañía californiana *OpenAI*, GPT-2 [12] es el sucesor del modelo de 2018 conocido como GPT. Está basado en la arquitectura Transformer y entrenado con un dataset de más de 8 millones de páginas webs. Tiene un vocabulario con un tamaño superior a las 50000 palabras y está disponible en 4 tamaños:

- *Small*: 124 millones de parámetros
- *Medium*: 355 millones de parámetros
- *Large*: 774 millones de parámetros
- *XL*: 1.5 billones de parámetros

Su enfoque es el del *Language Modeling*, es decir, predecir la siguiente palabra, conociendo todo el texto anterior. A partir de esto, podemos utilizarlo principalmente para 4 tareas: generación aleatoria de texto, respuesta a preguntas, resumen de textos y traducción automática.

Para las anteriores tareas utilizará el mismo enfoque de predecir la siguiente palabra, por lo que si por ejemplo queremos darle un contexto y que responda una pregunta la interacción se desarrollaría de la siguiente forma:

“*Mi nombre es Sergio y mi apellido es Olivares. Tengo 22 años. P: ¿Cuál es mi nombre? R: Sergio P: ¿Cuál es mi apellido? R: Olivares P: ¿Cuántos años tengo? R:*”

Y al intentar completar el texto lo que haría sería responder a la pregunta si el modelo funcionara correctamente y se le hubiera hecho un entrenamiento posterior para ello.

Se encuentra únicamente disponible en inglés, aunque es posible encontrar en la red algunos modelos entrenados con dataset en múltiples idiomas, entre los que se encuentra el español.

### 3.3 Text-To-Text Transfer Transformer (T5)

Creado por un equipo de científicos de *Google Research*, el modelo T5 [13] nace tras realizar un estudio a gran escala para determinar qué técnicas de Transferencia de Aprendizaje tienen mejores resultados.

Junto a este modelo, se introduce un dataset abierto de preentrenamiento conocido como *Colossal Clean Crawled Corpus* (C4), el cual se trata de una versión depurada del dataset de *Common Crawl* con un tamaño superior a los 100 GB.

Este modelo reformula todas las posibles tareas de PLN en un formato unificado de texto a texto en el que la entrada y la salida son siempre cadenas de texto. De esta forma, se usa el mismo modelo, función de pérdidas y parámetros para cualquier tarea de procesamiento de lenguaje, incluyendo traducción automática, análisis de sentimientos o respuesta a preguntas.

Este modelo, como ya se ha comentado, está basado en la arquitectura *Transformer*. La estructura base que se utiliza para el entrenamiento consiste de un codificador y un decodificador, cada uno con 12 bloques (cada bloque comprende una capa de *Self-Attention*, una capa opcional de *Encoder-Decoder Attention* y una red neuronal pre-alimentada). Manipula un total de 220 millones de parámetros.

Se utiliza un vocabulario de 32000 palabras, que cubre principalmente el inglés, pero también contiene alemán, francés y rumano. Por lo tanto, si quisiéramos hacer uso del modelo T5 en otro idioma, no tendríamos más remedio que descargar la versión multilingüe, conocida como mT5, y que cubre más de 100 idiomas.

Este modelo es de naturaleza multitarea, es decir, que ha sido entrenado para poder desarrollar tareas de diferente tipo con el mismo modelo, únicamente habría que indicarle cuál queremos con un prefijo. De esta forma, hay ciertas tareas para las que este modelo ya está entrenado y que podríamos usar sin tener que entrenarlo de nuevo. La enumeración y descripción de estas tareas se muestra en la Tabla 3.1.

Para otro tipo de tareas más específicas (conocidas como *downstream tasks*), se debería hacer uso de la Transferencia de Aprendizaje, de forma que a partir del conocimiento que este modelo ya tiene de todas las tareas anteriores, pudiera llevar a cabo cualquier otra tarea específica.

Con respecto al tamaño del modelo, T5 está disponible en diferentes formatos:

- *Base*: 220 millones de parámetros
- *Small*: 60 millones de parámetros
- *Large*: 770 millones de parámetros
- *3B*: 2.8 billones de parámetros
- *11B*: 11 billones de parámetros

**Tabla 3.1** Descripción de las tareas para las que está entrenado el modelo base de T5 [14].

Tarea	Descripción
CoLA	Determinar si una oración es gramaticalmente correcta
RTE	Determinar si una afirmación se puede deducir de una oración
MNLI	Determinar para una hipótesis y una premisa si se contradicen, si es así cuál, o si ninguna
MRPC	Determinar si dos oraciones son semánticamente equivalentes
QNLI	Determinar si la respuesta a una pregunta se puede deducir de una posible respuesta
QQP	Determinar si dos preguntas son semánticamente equivalentes
SST2	Determinar si el sentimiento que desprende una oración es positivo o negativo
STSB	Clasificar el sentimiento que desprende una oración en una escala de 1 a 5
CB	Determinar para una hipótesis y una premisa si se contradicen (binario)
COPA	Determinar para una pregunta, premisa y 2 opciones cual es la opción correcta
MultiRc	Determinar para una pregunta, un párrafo de texto y una posible respuesta, si la respuesta es correcta
WiC	Determinar para un par de oraciones y una palabra presente en ambas si la palabra tiene el mismo significado en las dos oraciones
WSC/DPR	Predecir para un pronombre ambiguo a qué palabra se refiere
Summarization	Resumir un texto
SQuAD	Responder a una pregunta para un contexto dado
WMT1	Traducir del inglés al alemán
WMT2	Traducir del inglés al francés
NQ	Respuesta a respuestas sin acceso a búsqueda externa

### 3.4 Bidirectional Encoder Representations from Transformers (BERT)

Este modelo es también diseñado por un equipo de científicos de *Google Research*, como el T5. Se conoce como BERT [15] y su principal característica es que permite un entrenamiento bidireccional aplicado a Procesamiento de Lenguaje natural. No sólo lee las secuencias de izquierda a derecha, sino también al contrario al mismo tiempo; lo cual permite una mayor comprensión del contexto por parte del modelo [16]. Así, mientras el GPT-2 sólo es capaz de mirar al texto anterior a la parte del texto que va a completar, BERT puede mirar al texto completo.

Ha sido entrenado con varios corpus que suman más de tres mil millones de palabras. El modelo original está disponible en dos formas:

- *Base*: 110 millones de parámetros. Usa una estructura Transformer con 12 codificadores.
- *Large*: 340 millones de parámetros. En este caso la estructura Transformer cuenta con 24 codificadores.

El modelo fue originalmente entrenado para tareas de "predicción de la próxima oración" (*Next Sentence Prediction*), que consiste en que recibía un par de oraciones y aprendía a detectar si la segunda era una continuación de la primera. No obstante, tras un entrenamiento posterior, es capaz de ser utilizado en tareas como análisis de sentimientos, resolución de preguntas o reconocimiento de entidades nombradas.

## 4 Aplicación en la Cobertura de Eventos Electorales Municipales

---

Este capítulo se centra en el proyecto que se ha llevado a cabo para aplicar las técnicas de Inteligencia Artificial para la redacción automática de textos, las cuales han sido introducidas en capítulos anteriores. Se comenzará presentando el ámbito de uso de la aplicación y las motivaciones por las que la consecución de este proyecto puede ser de utilidad en el mismo, para después describir los componentes de la aplicación final y cómo ha sido el proceso de desarrollo de ambas.

### 4.1 Motivación y Antecedentes

En este apartado procederemos a detallar algunos aspectos que conforman la motivación que ha empujado al desarrollo de este proyecto y a la herramienta usada para aplicarlo.

#### 4.1.1 Generación automática de textos periodísticos

El periodismo y la tecnología han convivido de manera satisfactoria en las últimas décadas. Esto se debe a la necesidad del primero, por su propia definición, de hacer la información lo más accesible y sencilla posible para sus lectores; y a la capacidad de la segunda para facilitar esto sin aumentar los costes. Así, era evidente que los últimos avances de la tecnología, encabezados por la Inteligencia Artificial, llegarían tarde o temprano al arte periodístico.

La tarea en la que nos centraremos es la redacción automática de textos periodísticos, cuyo objetivo es la producción de los mismos utilizando las ventajas de la automatización e instantaneidad que nos ofrecen los ordenadores. Esto no quiere decir que carezca de supervisión o intervención humana, la figura humana siempre deberá permanecer, ya sea estando presente en el diseño inicial del sistema, en el entrenamiento posterior del mismo o en la evaluación de la ejecución final [17].

Por tanto, el término más adecuado para definir al campo en el que se ubicará a este proyecto es el definido por el periodista Matt Carlson en [18]: **periodismo automatizado** (en inglés, *automated journalism*), cuya definición es la siguiente:

"Se define como aquella producción periodística en la que se transforman datos en textos elaborados a partir de algoritmos de Inteligencia Artificial con poca intervención humana más allá de la programación inicial del sistema."

Actualmente, existen ya varios medios nacionales e internacionales que han hecho uso de estas técnicas. Esto les ha permitido incrementar su productividad (más artículos en menos tiempo) y su diversidad (artículos más variados y específicos) [17]. Algunos de estos medios son *The Washington Post*, *Bloomberg* o *Associated Press* más allá de nuestras fronteras; y la Agencia EFE, el Diario Sport o El Confidencial en nuestro país.

#### 4.1.2 Eventos Electorales Municipales

Cada 4 años en nuestro país los españoles son llamados a las urnas para decidir los concejales que compondrán los Ayuntamientos de sus municipios. Se trata de un evento de especial trascendencia, pues las personas

que salgan electas de esas votaciones son las que tomarán las decisiones que afectan más a los ciudadanos en su día a día. Por tanto, es el deber del periodismo el informar adecuadamente de los resultados de estas elecciones en cuanto se empiezan a escutar los votos.

Nuestro estado se compone, según los datos del Ministerio del Interior en 2019, de 8131 municipios. Es lógico pensar que es imposible que un equipo humano pueda elaborar un artículo para cada uno de estos municipios a tiempo real mientras avanza el escrutinio. Además, sería un trabajo muy repetitivo y tedioso. Aquí es donde encontramos el objeto de nuestro proyecto: un generador de artículos periodísticos sobre cada uno de los municipios que vayan actualizándose a la vez que el conteo de votos.

Esto permitiría que hasta en el pueblo más pequeño cualquier persona pudiera tener disponible en cualquier momento un texto elaborado que poder leer sobre los resultados de los municipios, más allá que tablas y gráficos de difícil comprensión.

Esta aplicación que se comenta es la que llevaron a cabo el periódico El Confidencial y la empresa Narrativa con las elecciones municipales de 2019. Se muestran capturas de uno de estos artículos generados por Inteligencia Artificial en la Figura 4.1.

#### 4.1.3 Asistentes Virtuales

Los asistentes virtuales son herramientas diseñadas para mejorar la vida de las personas y proporcionar información rápida, veraz y al alcance de la mano. Llevan muchos años en nuestras vidas (*Siri* [20] fue lanzado en 2011, *Alexa* [21] en 2014 y *Google Assistant* [22] en 2016), pero ha sido recientemente con los altavoces inteligentes cuando han aumentado su popularidad y uso.

Además, tanto Amazon como Google han abierto sus asistentes en los últimos años, de forma que desarrolladores externos pueden crear subasistentes propios. Existen ya aplicaciones de asistente de voz de Uber, Domino's Pizza o NBC News. Un caso de ejemplo de aquí en España fue cuando el programa de televisión "Operación Triunfo" lanzó uno propio para *Google Assistant*, con el que podías interactuar para que te contara las últimas novedades dentro de la academia musical [23].

Con respecto a nuestro ámbito de aplicación, no existen precedentes conocidos en nuestro país de algo así. Por tanto, es este punto el que va a caracterizar a nuestro proyecto y lo que lo hace diferente a todo lo que se ha hecho hasta ahora. Partimos de la misma motivación que la aplicación que llevaron a cabo El Confidencial y Narrativa, pero yendo más allá y dándole otro enfoque a la utilidad de la redacción automática de textos periodísticos.

Imaginemos a una persona que quiere enterarse mientras avanza el escrutinio de quién va ganando en las elecciones de su municipio de apenas 200 habitantes. En la televisión hablarán de los resultados de capitales y ciudades importantes antes que de los de su pueblo. En los periódicos locales, aún teniendo suerte y pongamos que deciden publicar un artículo sobre su municipio, no podrá leerlo hasta el día siguiente. Las aplicaciones oficiales publicadas por el Ministerio del Interior para consultar los datos pueden ser muy engorrosas y complicadas. Con la aplicación que desglosaremos a continuación, sólo tendría que preguntárselo a su altavoz inteligente o a la aplicación de su asistente virtual en su teléfono y le contestaría en cuestión de segundos con los datos más actualizados.



**Figura 4.1** Captura del artículo publicado por El Confidencial sobre los resultados de las elecciones municipales de 2019 en el municipio de Puebla de Sanabria (Zamora), generado por IA [19].

## 4.2 Funcionalidad

Se trata de una aplicación de asistente de voz, en este caso *Google Assistant*, para proporcionar información a tiempo real sobre los resultados de las elecciones municipales de 2023, municipio a municipio, mientras avanza el escrutinio. Los usuarios dialogarán con un subasistente, cuyo nombre es "Zuri", al que podrán realizar peticiones a partir de naturales comandos de voz.

El subasistente de voz será capaz de:

- Informar de forma resumida sobre los resultados de un municipio concreto con el escrutinio actual
- Informar de forma extensa sobre los resultados de un municipio concreto con el escrutinio actual
- Proporcionar los resultados de forma visual para un municipio concreto con el escrutinio actual
- Responder a preguntas específicas sobre los resultados de un municipio concreto con el escrutinio actual

### 4.3 Componentes

A continuación se hará una descripción breve de los componentes de la aplicación objeto del proyecto, de forma que el lector pueda tener una visión global. Será en capítulos posteriores donde se detallarán estos componentes junto a la explicación del proceso de desarrollo de los mismos.

La aplicación se compone de múltiples elementos que residen en Internet, algunos con autoría propia y otros de organizaciones externas. Un diagrama de los componentes se puede apreciar en la Figura 4.2.

Se pueden observar en total tres grandes servidores que se comunican entre sí y gestionan las distintas peticiones que los usuarios realizan a través de una Interfaz de Usuario por voz, encarnada en el asistente virtual de *Google Assistant*. Asimismo, hay un servidor redireccionador que permite reenviar las peticiones y respuestas del servidor de la aplicación al de *Google*. Por último, existe una base de datos donde residen los resultados de las elecciones para cada uno de los municipios disponibles. La mayoría de peticiones y respuestas se hacen a través del protocolo HTTP sobre una capa de transporte segura SSL/TLS.

En primer lugar, el **servidor de aplicación** es de autoría propia y es el núcleo más importante del proyecto. Por un lado, se conecta a la base de datos para obtener los mismos y procesarlos, con el objetivo de que tengan un formato que acepte el generador de texto. Las peticiones a los generadores de texto las realiza al servidor de *HuggingFace*, ya que estos residen allí. Así, los resultados de estas peticiones los envía al servidor de *Google* elevando un servidor local que es redirigido a Internet gracias a la herramienta *ngrok* [24], que es capaz de crear túneles seguros que puedan ser accedidos desde fuera de Internet.

Con respecto a la **base de datos**, aunque en el diagrama de componentes aparece como un elemento externo al servidor de la aplicación, en la implementación final, que será explicada en apartados posteriores, se verá que no es así. Idealmente, esta base de datos debería ser externa y residir en algún servidor del Ministerio del Interior, el cual fuera accesible a la vez que se fueran contando los votos, y que de esta forma los usuarios de nuestra aplicación pudieran acceder a esta información en tiempo real. Al no encontrarnos en elecciones, y no existir este servidor, se decide en este proyecto utilizar un sustituto de prueba.

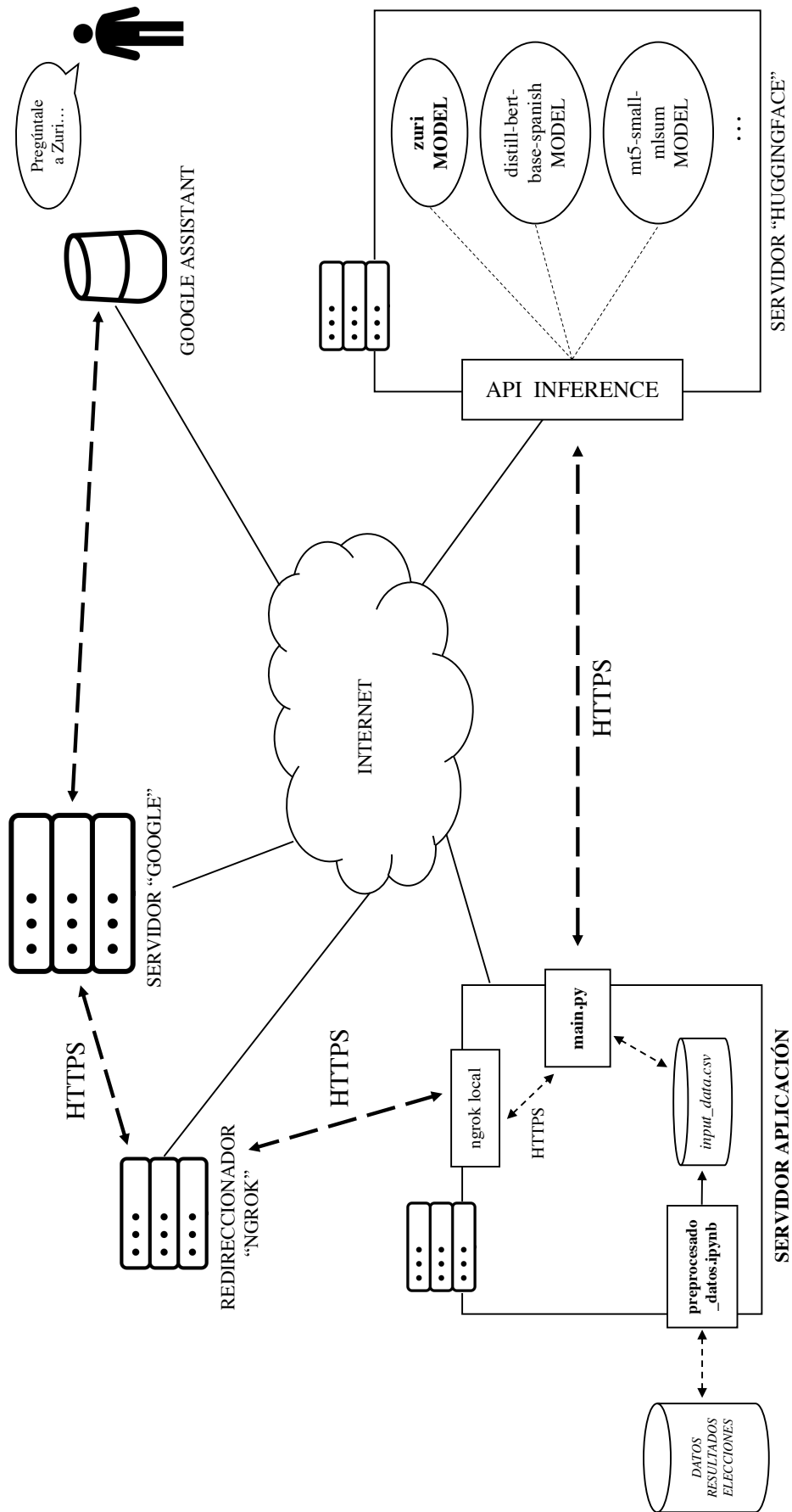
*HuggingFace* [25] es una plataforma online con miles de modelos pre-entrenados y datasets disponibles. Se dispone de una herramienta, *API INFERENCE*, que te permite realizar solicitudes y recibir respuestas de estos modelos a unas velocidades hasta 100 veces superior a las que podrías hacerlo si el modelo residiera en tu propio ordenador. En esta plataforma, residen los modelos "mt5-small-mlsum" [26] y "distill-bert-base-spanish" [27], los cuales han sido entrenados por autores externos y de los que nos aprovecharemos para realizar tareas de Summarization y SQuAD (ver Tabla 3.1). También se encuentra en este servidor el modelo de autoría propia de este proyecto, "zuri", que utilizaremos para la generación de textos periodísticos a partir de datos de elecciones.

Finalmente, el **servidor de Google** maneja las peticiones del usuario y las procesa según se ha indicado en la plataforma *DialogFlow* [28], la cual permite crear aplicaciones externas de *Google Assistant* a través de una interfaz visual. Es en ella dónde indicamos que cuando el usuario haga determinadas preguntas a "Zuri", se realicen ciertas peticiones al servidor *ngrok*, el cual las redirigirá al servidor de nuestra aplicación. Allí podemos también personalizar la voz de nuestro subasistente o hacer pruebas con él. Podemos ver una captura de la plataforma en la Figura 4.3.

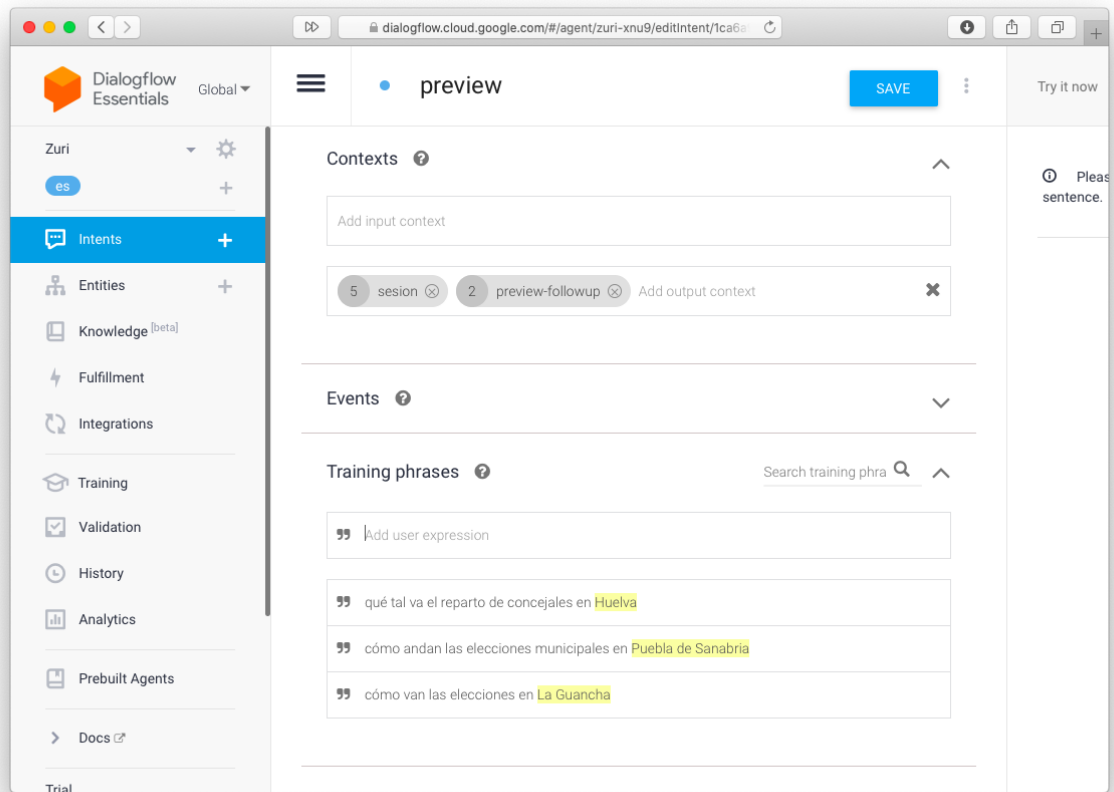
#### 4.3.1 Bibliotecas y frameworks utilizados

Todo el código que se ha desarrollado en este proyecto para la implementación de los anteriores componentes ha sido escrito en Python. Asimismo, se ha hecho uso de varias librerías en este lenguaje para facilitar y hacer más eficiente el proceso de desarrollo del mismo. Se describen las más importantes brevemente a continuación:

- *Pandas* [29]: herramienta para procesamiento, manipulación y análisis de datos, a través de objetos denominados *DataFrame*. Está construido sobre dos librerías base de Python: *matplotlib* [30] y *NumPy* [31].
- *PyTorch* [32]: diseñado por el equipo de investigación de IA de *Facebook*, es una librería de código abierto destinada a aplicaciones de *Machine Learning*. Sus características más destacadas son la computación tensorial a través de unidades de procesamiento gráfico y el uso de redes neuronales profundas.
- *Transformers* [33]: proporciona una abstracción de propósito general destinado a la Generación de Lenguaje Natural y Comprensión de Lenguaje Natural, independientemente del modelo o idioma. Es totalmente operable con *HuggingFace*.



**Figura 4.2** Diagrama de componentes de la aplicación en funcionamiento, así como las distintas comunicaciones que se producen entre sí. En negrita aquellos componentes que pertenecen a los autores de este proyecto..



**Figura 4.3** Captura del *intent* "preview" de la aplicación del subasistente virtual de Google Assistant, "Zuri", en la plataforma *DialogFlow*.

- *NLTK* [34]: es un conjunto de librerías enfocadas al Procesamiento de Lenguaje Natural en Python. Incluye métodos para clasificación, tokenización, buscador de lexemas, etiquetado, análisis sintáctico, etc.
- *Sentencepiece* [35]: es un tokenizador de texto diseñado específicamente para generadores de texto basados en redes neuronales. No necesita supervisión.
- *Flask Assistant* [36]: se trata de una extensión de la librería *Flask* (un microframework web para Python), que permite fácilmente crear asistentes virtuales comunicándose con *DialogFlow* a través de webhooks.
- *BeautifulSoup* [37]: biblioteca que permite manipular archivos XML y HTML, de forma que se pueda obtener partes específicas de dichos códigos de forma sencilla.
- *urllib* [38]: librería para trabajar con enlaces URLs - procesarlos, enviar solicitudes HTTP y recibir la respuesta, etc.
- *openpyxl* [39]: paquete para manipular archivos Excel.
- *dhondt* [40]: no se trataría de una librería como tal, sino una clase de Python que realiza el cálculo de distribución de escaños dado un reparto de votos, basándose en la ley d'Hondt.

#### 4.4 Proceso de diseño e implementación

En el apartado que comienza con estas líneas se procede a hacer una explicación detallada de los pasos que se han seguido a la hora de implementar en código los componentes propios junto con una explicación más detallada de los mismos, así como argumentaciones de por qué se han tomado ciertas decisiones y no otras y los problemas que se hayan podido tener. Se ha dividido en tres subapartados correspondientes a cada una de las siguientes cuestiones: cómo ha sido el proceso para crear un generador de texto a partir de un modelo pre-entrenado, así como para incorporarlo a una aplicación escrita en Python junto a otros modelos



ya existentes (**Generación de texto**), cómo la aplicación obtiene los datos y los procesa para transformarlos en una entrada leíble por el generador de textos (**Obtención y Procesado de Datos**) y, por último, cómo se consigue que todo esto pueda ser utilizado en una Interfaz de usuario por voz (**Interacción del Usuario con el Asistente Virtual**).

#### 4.4.1 Generación de Texto

Este proyecto comienza casi un año antes de su publicación con un proceso de investigación y estudio de las técnicas existentes en el campo de Procesamiento de Lenguaje Natural y Generación de Lenguajes Naturales. Se inicia entendiendo cómo se consigue que un ordenador sea capaz de comprender textos naturales, para después pasar a cómo puede ser capaz de generarlos.

Primero se entra en contacto con el modelo explicado en el apartado 3.2: GPT-2. Se empieza con él ya que, como indicamos, es de los modelos más avanzados que existen hoy en día y ciertamente produce resultados extraordinarios. No obstante, tras realizar pruebas con él (por ejemplo, se implementa un generador de tweets del expresidente de los EEUU, Donald Trump), se observa que las posibilidades de generación condicionada de este modelo son muy limitadas: únicamente podías indicar unas ciertas palabras de comienzo del texto para que el generador lo completara, y a lo máximo podías indicar algunos *topics* para guiarlo, pero nada más. De esta forma, se entendía que este modelo carecía de utilidad para nuestro proyecto de generación de textos a partir de datos muy concretos. No interesaba un modelo que genere texto, aunque con sentido, pero inexacto, si el PSOE había ganado con un 33.54 % de los votos, necesitábamos que el modelo lo indicara concretamente.

En esta búsqueda de un sistema que fuera capaz de transformar datos exactos en texto elaborado nos encontramos con un artículo del programador Mathew Alexander [41] en el que, ajustando el modelo de Google T5 (también explicado, en este caso en el apartado 3.3) conseguía convertir filas de datos en simples frases. Para entrenar al modelo, utilizaba un dataset del concurso lanzado en 2017 conocido como WebNLG [42], que consistía en retar a diversos programadores a que consiguieran diseñar un modelo que transformara datos en textos y compararlos para ver quién obtenía mejores resultados. Lo que consigue Alexander en su publicación es que introduciendo la siguiente entrada de ejemplo al modelo:

```
<mtriple> Trane | foundingDate | 1913-01-01    </mtriple>
<mtriple> Trane | location | Ireland            </mtriple>
<mtriple> Trane | foundingPlace | La_Crosse,_Wisconsin </mtriple>
<mtriple> Trane | numberOfEmployees | 29000    </mtriple>
```

Se obtenga de respuesta este texto:

*Trane, which was founded on January 1st 1913 in La Crosse, Wisconsin, is based in Ireland. It has 29,000 employees.*

Así, se toma este proyecto como base para el nuestro, pues compartía muchos de sus requisitos. En primer lugar, se tenía que decidir cómo serán los datos de entrada del modelo, para que los transforme en un artículo que informe sobre los resultados de las elecciones. Tras realizar múltiples pruebas para entender cómo ajustar el modelo y la entrada de texto para que funcione perfectamente, llegamos al formato de entrada definitivo para nuestra aplicación que se compone de los siguientes elementos:

- Prefijo indicador de tarea
- Nombre del municipio
- Porcentaje actual de escrutinio
- Por cada uno de los partidos que han obtenido votos...
  - Nombre del partido
  - Porcentaje de voto
  - Número de escaños obtenido
  - Diferencia de escaños con respecto a las anteriores elecciones

Estos elementos tendrán el siguiente formato:

```
texto: <municipio> | <escrutinio> = <partido1> | <voto1> | <escaños1> | <
diferencial1> && <partido2> | <voto2> | <escaños2> | <diferencia2> && ...
```

Un ejemplo para unos resultados en el municipio de Robleda-Cervantes con un escrutinio de 78.32 %, el PSOE ganando con un 70.12 % de los votos, 8 escaños y aumentando en 3 escaños con respecto a las anteriores elecciones; mientras que se encuentra el PP obteniendo un 30.88 % de los votos, 3 escaños y 3 menos que los anteriores comicios sería:

```
texto: Robleda-Cervantes | 78.32 = PSOE | 70.12 | 8 | 3 más && PP | 30.88 | 3 |
3 menos
```

No obstante, queda una parte muy importante: obtener textos elaborados con los que entrenar al modelo y que aprenda a hacer lo que buscamos. Gracias a la Transferencia de Aprendizaje (ver apartado 2.3) podemos hacerlo con una cantidad de datos inferior a lo necesario en una situación normal, ya que nos aprovechamos de un modelo que ya ha sido entrenado para entender el español. Sólo se le necesitaba enseñar cómo debe elaborar un texto periodístico con los datos que se le introducen en la entrada.

Aquí es donde entra el trabajo realizado por el medio El Confidencial que explicamos en el apartado 4.1.2. Al tener una gran cantidad de artículos elaborados para las elecciones municipales de 2019 para casi todos los municipios españoles, se decide utilizar parte de ellos para que nuestro modelo "zuri" aprenda.

Para ello necesitábamos implementar un sistema de *Web Scraping*, es decir, de escaneo de páginas webs. Esto se debe a que se tratan de más de 2000 artículos que copiar, y hacerlo de forma manual era inviable. El sistema a diseñar debía ser capaz de buscar en *Google* el artículo de El Confidencial de cada municipio, acceder al mismo, y coger la parte que nos interesaba. Esto lo conseguimos codificándolo en un notebook de Python y gracias a la ayuda de la herramienta *ScraperApi* [43].

El problema del *Web Scraping* es que la mayoría de páginas webs tienen implementados sistemas para evitar ataques de Denegación de Servicio (*DoS*), de forma que cuando accedes muchas veces seguidas te deniega el acceso o te pide que resuelvas un código Captcha. *ScraperApi* te permite que puedas acceder fácilmente a múltiples elementos de una web sin que esto pase, con distintas técnicas como rotaciones de IPs o modificaciones en las cabeceras HTTP. El uso es muy sencillo: sólo hay que añadirle un prefijo a la URL que queremos entrar, de forma que el acceso se haga a través de los servidores de *ScraperApi* y no los propios.

Una vez solucionado este problema, se implementa un código que, en primer lugar, se le introduce una lista de los nombres de todos los municipios de España. Para cada uno de ellos, el sistema busca en *Google* si existe un artículo de las elecciones municipales de 2019 de El Confidencial y, si es así, descarga el código HTML de esa web y recoge el primer párrafo del artículo, que era el que nos interesaba. Obtiene el texto y lo guarda en una base de datos en formato CSV. El código se puede consultar en el Apéndice A como A.1.

Tras este proceso, se obtuvo un dataset de más de 2500 filas. Para generar las entradas correspondientes a esas salidas en el dataset, se utilizó el código que se explicará en el siguiente apartado.

Se realiza el entrenamiento del modelo con este dataset que hemos diseñado y obtenemos finalmente el modelo "zuri" en funcionamiento. Los resultados son muy satisfactorios y las características finales del mismo se describen en la Tabla 4.1. Un ejemplo de generación de nuestro modelo utilizando los datos ficticios anteriores de Robleda-Cervantes se muestra a continuación. Para la siguiente entrada:

```
generate("texto: Robleda-Cervantes | 78.32 = PSOE | 70.12 | 8 | 3 más && PP |
30.88 | 3 | 3 menos")
```

Obtenemos la siguiente salida:

*Con un 78.32 % de los votos escrutados, el PSOE triunfaría en las elecciones municipales de Robleda-Cervantes con ocho concejales, lo que se traduciría en un mejor resultado en la región que en los comicios anteriores, pues se traduce en tres asientos más. La organización de Ferraz habría logrado un 70,12 % de los votos, mientras que en segundo lugar aparecería el PP, que con tres ediles, lograría tres menos que en las pasadas votaciones en esta localidad.*

Hay que destacar varias cosas. Por un lado, que detecta perfectamente los datos introducidos en la entrada y los coloca en el texto de forma natural y con cierto lenguaje periodístico, incluyendo lo referido a la

**Tabla 4.1** Características del modelo de generación de texto propio "zuri".

Modelo pre-entrenado usado	"mt5-small-mlsum" de Leonardo Ignacio Córdoba (basado en mT5 de <i>Google</i> )
Tokenizer usado	mT5 Tokenizer de <i>Google</i>
Tarea para la que ha sido entrenado	<i>DataToText</i> enfocado a resultados electorales municipales
Idioma	Español
Tamaño de los lotes de entrenamiento ( <i>batch size</i> )	2 muestras
Número de secuencias de entrenamiento ( <i>number of epochs</i> )	4
Alias en <i>HuggingFace</i>	"olirods/zuri"

comparación con años anteriores, pues con una entrada de "3 más", lo redacta como "lo que se traduciría en un mejor resultado en la región que en los comicios anteriores, pues se traduce en tres asientos más". Además, de todos los datos, sabe intuir que lo más importante es el ganador y es en lo que más se centra el artículo. Todo esto es posible tras el entrenamiento con más de 2500 filas, de forma que el modelo ha ido aprendiendo cómo debía de ir colocando todos estos datos y de qué forma.

Para entrenarlo, se utilizó la plataforma *Google Colaboratory*, pues permite la ejecución de notebooks en Python en GPUs externas de gran capacidad. El código se puede consultar en el Apéndice A como A.2.

El modelo anterior reúne las características objeto del proyecto: conseguir generación de textos periódicos a partir de datos electorales. Sin embargo, la aplicación no se queda ahí e incorpora dos funciones complementarias más: la capacidad de hacer un resumen corto de los resultados y la habilidad de responder preguntas específicas para esos resultados. Estas dos funciones se consiguen mediante otros dos modelos que, eso sí, no son de autoría propia.

El primero de ellos es el que se ha usado como base para el modelo "zuri", el **modelo "mt5-small-mlsum"** [26], el cual se ha entrenado específicamente para hacer tareas de Summarization en español. Lo usaremos para poder generar titulares o resúmenes del artículo completo, lo cual será útil para informar brevemente al usuario de forma rápida en vez de soltarle un texto de demasiada longitud. Lo veremos en próximos apartados.

Así, si le introducimos el artículo de Robleda-Cervantes al modelo:

```
generate("summarize: Con un 78.32 de los votos escrutados, el PSOE triunfaría en las elecciones municipales de Robleda-Cervantes con ocho concejales, lo que se traduciría en un mejor resultado en la región que en los comicios anteriores, pues se traduce en tres asientos más. La organización de Ferraz habría logrado un 70,12 de los votos, mientras que en segundo lugar aparecería el PP, que con tres ediles, lograría tres menos que en las pasadas votaciones en esta localidad.")
```

La salida es:

*El PSOE triunfaría en las elecciones municipales de Robleda-Cervantes con ocho concejales*

Conseguimos un titular corto pero con la información más importante del texto: el ganador de las elecciones. Las características del modelo se muestran en la Tabla 4.2.

El otro modelo externo que utilizamos se conoce como "**distill-bert-base-spanish-wwm-cased-finetuned-spa-squad2-es**" [27] y se ha entrenado para tareas de respuesta a preguntas en español. En este caso lo usamos para que el usuario pueda realizar preguntas concretas cortas sin tener que leer el artículo completo. El modelo funciona buscando la información en el texto que se le introduce, que en este caso será el artículo obtenido con el modelo "zuri". Vamos a ver algunos ejemplos.

Si únicamente quisiéramos saber los escaños específicos que ha obtenido un partido específico, en este caso el Partido Popular, ejecutaríamos la siguiente entrada:

```
generate(
{
```

**Tabla 4.2** Características del modelo de generación de texto externo "mt5-small-mlsum".

Modelo pre-entrenado usado	mT5 de <i>Google</i>
Tokenizer usado	mT5 Tokenizer de <i>Google</i>
Tarea para la que ha sido entrenado	Summarization
Idioma	Español
Tamaño de los lotes de entrenamiento ( <i>batch size</i> )	4 muestras
Número de secuencias de entrenamiento ( <i>number of epochs</i> )	10
Alias en <i>HuggingFace</i>	"LeoCordoba/mt5-small-mlsum"

```

'question': '¿Cuántos escaños obtendría el PP?',
'context': 'Con un 78.32 de los votos escrutados, el PSOE triunfaría en
las elecciones municipales de Robleda-Cervantes con ocho concejales
, lo que se traduciría en un mejor resultado en la región que en los
comicios anteriores, pues se traduce en tres asientos más. La
organización de Ferraz habría logrado un 70,12 de los votos,
mientras que en segundo lugar aparecería el PP, que con tres ediles,
lograría tres menos que en las pasadas votaciones en esta localidad
.'
}
)

```

Se genera la siguiente salida:

*tres ediles*

Se puede observar que el resultado es certero y que además no es necesario utilizar palabras exactas. En este ejemplo hemos visto cómo hemos usado la palabra "escaños" para referirnos a los concejales obtenidos por la formación popular, aunque en el texto aparecían como ediles.

Otro ejemplo podría ser preguntar cuál es el porcentaje exacto de escrutinio, para saber si los resultados son definitivos o no. Se haría con la siguiente entrada:

```

generate(
{
'question': '¿Qué porcentaje hay de escrutinio?',
'context': 'Con un 78.32 de los votos escrutados, el PSOE triunfaría en
las elecciones municipales de Robleda-Cervantes con ocho concejales
, lo que se traduciría en un mejor resultado en la región que en los
comicios anteriores, pues se traduce en tres asientos más. La
organización de Ferraz habría logrado un 70,12 de los votos,
mientras que en segundo lugar aparecería el PP, que con tres ediles,
lograría tres menos que en las pasadas votaciones en esta localidad
.'
}
)

```

Y obtenemos esta respuesta por parte del modelo:

*78.32%*

En próximos apartados, se verá como esto es de gran utilidad para el asistente de voz, ya que es un sistema que debe ser capaz de gestionar cualquier tipo de pregunta por parte del usuario. Una descripción breve de las características de este modelo se muestran en la Tabla 4.3.

**Tabla 4.3** Características del modelo de generación de texto externo "distill-bert-base-spanish".

Modelo pre-entrenado usado	Modelo BERT multilinguaje
Tokenizer usado	BERT Tokenizer
Tarea para la que ha sido entrenado	SQuAD
Idioma	Español
Tamaño de los lotes de entrenamiento ( <i>batch size</i> )	12 muestras
Número de secuencias de entrenamiento ( <i>number of epochs</i> )	5
Alias en <i>HuggingFace</i>	"mrm8488/distill-bert-base-spanish-wwm-cased-finetuned-spa-squad2-es"

### Acceso a los modelos

Una vez que teníamos todos los modelos se necesitaba:

1. Que fueran accesibles por Python
2. Que pudieran ser llamados durante la ejecución del código
3. Que la generación de texto no durase más de 10 segundos

La primera de ellas no tenía problema ya que este lenguaje es el propio de análisis de datos y PLN, por lo que todas las herramientas de las que disponíamos eran compatibles con Python. La segunda premisa se debía a que, al tratarse la aplicación de un asistente de voz, el sistema a diseñar sería del tipo cliente/servidor, en el que nuestro servidor estaría escuchando llamadas y respondiendo a ellas. Este punto tampoco generaba muchas dificultades.

No obstante, las limitaciones se encontraron con el tercero. Esta premisa se debe a que el sistema de *Google Assistant* tiene un límite de 5 segundos para que nuestro sistema externo responda a sus llamadas por razones de experiencia de usuario. Con algunas técnicas que explicaremos más adelante, conseguimos aumentar este límite en hasta 10 segundos.

En una primera etapa, se intenta realizar la instalación y ejecución de los modelos en el propio servidor de la aplicación mediante la librería *Transformers*. Sin embargo, además de que el levantamiento del servidor era muy lento, la propia generación de texto una vez arrancados los modelos podía durar entre 10 y 30 segundos, algo inviable y que incumplía la tercera limitación de las anteriores.

Por lo tanto, se acaba optando por la herramienta de la plataforma *HuggingFace* conocida como *Accelerated Inference API*. Dicho servicio te permite correr tus modelos en su servidor de alta capacidad y realizar simples peticiones HTTP para obtener los resultados. Tras aplicarlo, conseguimos obtener resultados de generaciones de texto de 3 a 10 segundos. Limitación superada.

#### 4.4.2 Obtención y Procesado de Datos

Como se explicó en el apartado 4.3, la obtención de datos es una simulación estática de la situación real, ya que no disponemos de un sistema que pueda emular al que se utilizaría en las elecciones municipales de 2023. Lo que haremos será tomar las hojas Excel que se encuentran disponibles en la página del Ministerio del Interior [44] con los datos de las elecciones de 2019 y las de 2015. Necesitaremos ambos años ya que necesitamos comparar los resultados de unas elecciones a otras para poder indicar la mejora o empeoramiento de escaños que han podido tener los partidos.

Supondremos que estas bases de datos son las que iríamos teniendo en tiempo real una vez que se fueran conociendo los datos. Se hará un procesado de las mismas para poder obtener los resultados en el formato de entrada explicado en la sección anterior.

Hay que aclarar que en estas hojas que nos ofrece el Gobierno sólo aparecen los votos totales que ha obtenido cada candidatura. Por lo tanto, el primer paso a tomar será poder convertir esos votos en escaños, siguiendo las indicaciones que señala la Ley Orgánica del Régimen Electoral General [45]:

- Cada municipio constituye una circunscripción única.
- El número de concejales a repartir en cada municipio será según indique la Tabla 4.4 y el censo total de habitantes del municipio.

- Los escaños se reparten mediante escrutinio proporcional plurinominal con listas cerradas utilizando la ley D'Hondt.
- La barrera electoral, es decir, el mínimo porcentaje que debe sacar un partido para que pueda participar en el reparto de concejales, será del 5%.

**Tabla 4.4** Número de concejales a repartir en un municipio según el número de habitantes en las elecciones municipales españolas [45].

Hasta 100 habitantes	3
De 101 a 250	5
De 251 a 1,000	7
De 1,001 a 2,000	9
De 2,001 a 5,000	11
De 5,001 a 10,000	13
De 10,001 a 20,000	17
De 20,001 a 50,000	21
De 50,001 a 100,000	25
De 100,001 en adelante	25 + 1 por cada 100,000 o fracción (1 más si par)

Por tanto, siguiendo esta norma y gracias a la librería `dhondt` convertimos los votos de cada uno de los partidos de cada municipio en sus correspondientes escaños tanto para los datos de las elecciones de 2019 como las de 2015. Una vez calculados, simplemente tenemos que calcular la diferencia entre los dos municipios para cada uno de los partidos y tendríamos la base de datos completa. Como se ha nombrado anteriormente, la obtendríamos en forma de un fichero CSV que residiría en el servidor de la aplicación y que el código en Python encargado de la gestión de llamadas iría leyendo para responder a las mismas. El código se puede consultar en el Apéndice A como A.3.

Así, conseguimos transformar unas entradas sin procesar como las que se muestran a continuación. La del proceso electoral actual:

comunidad	provincia	municipio	poblacion	escrutinio	mesas	censo	votos_total
Andalucía	Almería	Laroya	169	66.34	1	185	165
votos_validos	votos_cand	votos_blanco	votos_nulos	PSOE	PP	Cs	
161	161	0	4	2	110	46	

Y la del proceso electoral previo:

comunidad	provincia	municipio	poblacion	mesas	censo	votos_total
Andalucía	Almería	Laroya	170	1	155	141
votos_validos	votos_cand	votos_blanco	votos_nulos	PP	PSOE	Cs
140	139	1	1	98	38	0

Se convertirían en la siguiente entrada para nuestro modelo "zuri" con el formato adecuado:

```
texto: Laroya | 66.34 = PP | 68.32 | 4 | 0 && Cs | 28.57 | 1 | <null> && PSOE |  
1.24 | 0 | 1 menos
```

#### 4.4.3 Interacción del Usuario con el Asistente Virtual

Utilizamos la plataforma *DialogFlow* para crear nuestra aplicación para el asistente de voz *Google Assistant*. En ella, podemos diseñar cómo funcionara nuestro subasistente de voz a base de definir múltiples *intents*. Cada uno de ellos representa una intención del usuario, y cómo nuestra aplicación responderá a esta intención. En nuestra aplicación existen cinco:

- "default": se lanza cuando se le da la bienvenida al usuario. El asistente de voz se presenta e indica qué preguntas se le pueden hacer.

- "preview": se lanza cuando el usuario quiere un breve titular o resumen de la situación actual de los resultados en un municipio. El asistente de voz se lo muestra y le indica si quiere ampliar la información. En caso afirmativo, lanzaría el *intent* "full". En caso negativo, lanzaría el *intent* "bye"
- "random": se lanza cuando el usuario lanza una pregunta concreta sobre los resultados de las elecciones, como "¿cuántos escaños obtendría el PP en Sevilla?" o "¿cuál es el porcentaje escrutado en Betanzos?". El asistente de voz se lo responde y le indica si quiere ampliar la información. En caso afirmativo, lanzaría el *intent* "full". En caso negativo, lanzaría el *intent* "bye"
- "full": el asistente de voz muestra el artículo completo junto a una tarjeta de información con el escudo del municipio concreto.
- "bye": el asistente de voz se despide del usuario.

Un diagrama de flujo del funcionamiento de la aplicación se muestra en la Figura 4.4. En la plataforma de *DialogFlow*, en cada uno de ellos, se escriben varias frases de entrenamiento que son las que consideramos que el usuario utilizaría para indicar esa intención concreta que tiene. Funcionan de forma que el sistema es capaz de analizarlas y después entender frases similares. En ellas además marcamos qué palabras concretas de las preguntas que hagan los usuarios queremos que sean enviadas a nuestra aplicación como parámetros. En la Figura 4.3 que se encuentra en apartados anteriores se puede apreciar parte de la configuración del *intent* "preview", donde hemos marcado el nombre del municipio en las frases de entrenamiento para que el sistema sepa dónde buscarlas dentro de la petición del usuario. En el caso de que el usuario no mencionara el nombre del municipio, algo indispensable para que sepamos de dónde recoger los datos, el propio asistente de voz lo pedirá. Esto se verá en funcionamiento en los ejemplos de validación del Capítulo 5.

El último ajuste que queda por comentar en la plataforma *DialogFlow* es el *Fulfillment*. Allí indicamos que cada vez que una *intent* sea lanzada, será gestionada a través de webhooks, es decir, de peticiones HTTP a través de Internet. Aquí es donde enlazamos nuestra aplicación, de forma que recibimos las peticiones de los usuarios, las pasamos a los modelos de generación de texto y enviamos las respuestas.

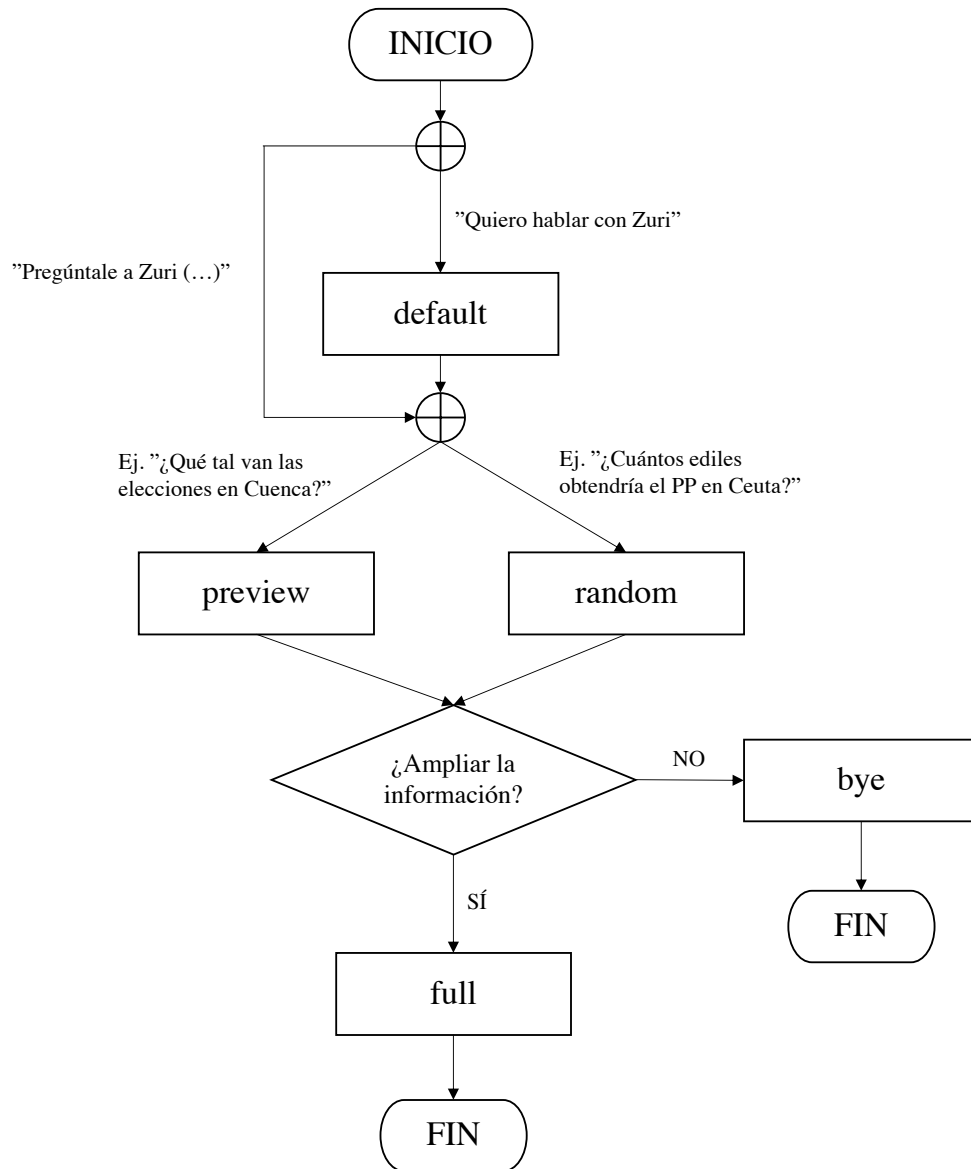
Para ello, utilizamos dos herramientas o librerías: *ngrok* y *Flask Assistant*. La primera nos permite crear un túnel seguro de forma que el servidor de nuestra aplicación sea accesible desde Internet mediante un enlace web que nos proporciona a arrancar la aplicación (este es el que indicaremos en la plataforma de *DialogFlow*). Simplemente debemos indicarle el puerto que queremos que esté abierto de nuestra máquina local (en nuestro caso el 8000) y se inicia mostrando una pantalla en el terminal como la que se ve en la Figura 4.5.

La segunda es la que nos permite interactuar con la aplicación de *Google Assistant* levantando un servidor que responda a sus llamadas HTTP. Definimos una serie de métodos que serán llamados cuando sus *intents* correspondientes sean lanzadas y estos tendrán como argumentos los parámetros que antes indicamos. Únicamente necesitamos llamar desde esos métodos a los modelos de generación de texto que residen en *HuggingFace* y las respuestas enviarlas de nuevo al asistente de voz.

El procedimiento que se sigue para llamar a los modelos de generación de texto es el siguiente:

1. La primera *intent* que necesitará la llamada a un modelo de generación de texto podrá ser "preview" (titular del artículo completo) o "random" (pregunta específica sobre el artículo completo). Ambas requieren del artículo completo para funcionar, por lo que se genera el mismo haciendo la llamada correspondiente al modelo "zuri" y guardándolo en la memoria.
2. Una vez tenemos el artículo completo, se procede a hacer la generación de texto del modelo correspondiente. Si el *intent* era "preview", se hace el resumen del artículo con el modelo "mt5-small-mlsum"; si era "random", se resuelve la pregunta utilizando el modelo "distill-bert-base-spanish".
3. Se envía la respuesta al asistente de voz
4. Si el usuario quisiera ampliar la información, le enviamos de forma instantánea el artículo completo que anteriormente habíamos guardado en memoria.

Como se puede predecir, esta doble llamada a los modelos para obtener la respuesta produce un aumento en el tiempo que tarda el sistema en elaborar el texto resultado. Esto acarrea problemas ya que *Google Assistant* tiene un límite de 5 segundos para recibir la respuesta del webhook, por lo que se tuvo que diseñar un método para aumentar este límite de forma manual además de acabar decidiendo que el acceso a los modelos fuera a través de *HuggingFace* como se explica en apartados anteriores.



**Figura 4.4** Diagrama de flujo del funcionamiento del subasistente de voz "Zuri" en función de las peticiones del usuario y las *intents* activadas. .

#### Gestión del tiempo de elaboración de los resultados

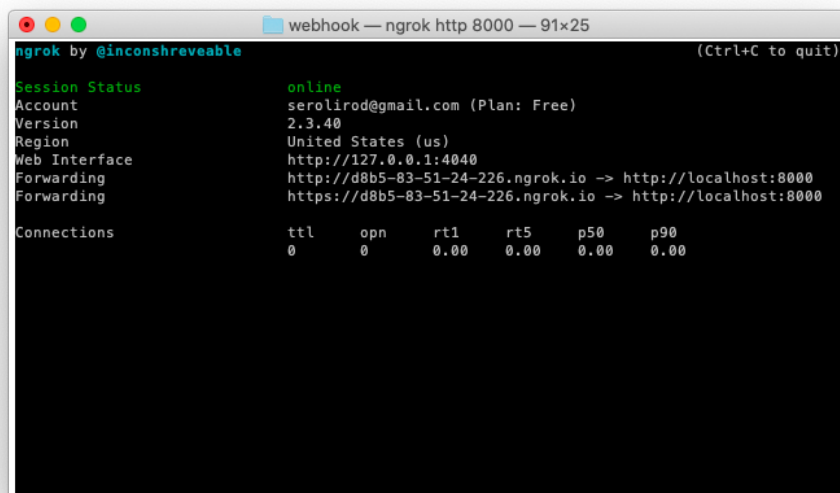
Aunque con el acceso a los modelos a través de *HuggingFace* reducimos el tiempo de espera para obtener resultados, era imposible garantizar que este tiempo fuera inferior a 5 segundos debido al tener que acceder a dos modelos para obtener una respuesta. Por lo tanto, se decide usar un método para intentar aumentar esta estricta ventana.

El código de nuestro servidor de Python se ejecutaba, como es usual, de forma secuencial. Esto producía que no podíamos responder al servidor de Google sin recibir respuesta del modelo. Por lo tanto, en primer lugar eliminamos esta limitación utilizando la librería *asyncio*, con la que procedemos a iniciar la llamada a los modelos de generación de texto en segundo plano, de forma que de mientras podamos hacer tiempo y responder a Google.

Por otro lado, para aumentar la ventana de tiempo, nos servimos de dos *intents* auxiliares: "random-followup" y "preview-followup". Estas dos *intents* serán llamadas por nuestra aplicación después de ser lanzada la *intent* principal, y nos permitirán "engañar" al servidor de Google aumentando en otros 5 segundos el margen de tiempo, de forma que en ese período ya sí nos habrá dado tiempo a enviar y recibir de Hugging Face la respuesta de los modelos.

El código completo de esta sección se puede consultar en el Apéndice A como A.4.





```
ngrok by @inconshreveable (Ctrl+C to quit)

Session Status      online
Account             serolirod@gmail.com (Plan: Free)
Version             2.3.40
Region              United States (us)
Web Interface       http://127.0.0.1:4040
Forwarding           http://d8b5-83-51-24-226.ngrok.io -> http://localhost:8000
                   https://d8b5-83-51-24-226.ngrok.io -> http://localhost:8000

Connections
  ttl    opn    rt1    rt5    p50    p90
   0      0     0.00   0.00   0.00   0.00
```

**Figura 4.5** Captura del redireccionador ngrok en ejecución en un terminal del servidor de la aplicación. .

En el próximo capítulo podremos ver la aplicación en acción. Se podrá observar cómo todo este complejo engranaje de componentes es totalmente invisible al usuario final, quien simplemente realiza peticiones y el asistente le responde con texto e imágenes.



## 5 Ejemplos de validación

A lo largo de este capítulo se mostrarán capturas de la aplicación en funcionamiento en la que se fundirán todos los componentes y aspectos que hemos visto en el capítulo anterior. Se describirán los aciertos y errores del sistema en tres casos y sus posibles orígenes. Se han elegido estos casos por su particularidad y para que puedan ser útiles en su comparación de forma que muestren las fortalezas y debilidades de esta aplicación.

Al tratarse de un subasistente dentro de *Google Assistant*, se podrá acceder a ella desde cualquier dispositivo compatible con este asistente de voz. Esto incluye los sistemas operativos móviles iOS y Android, pero también en altavoces y pantallas inteligentes como Google Home, Google Nest Hub, JBL Link Music o Polk Assist; e incluso está llegando a lugares como coches, televisiones o relojes inteligentes. En este caso, para los ejemplos de validación de este proyecto que se mostrarán a continuación se ha utilizado un iPad mini 2 corriendo iOS 12.4.9 y la aplicación Google Assistant que se encuentra en el App Store de Apple.

### 5.1 Datos de ejemplo utilizados

Para los casos que se describirán a continuación, tanto el servidor de aplicación como el redireccionador *ngrok*, así como los respectivos servidores de *HuggingFace* y Google estaban en funcionamiento. En el caso de la base de datos, se ha utilizado una versión de prueba reducida con datos introducidos manualmente, de los cuales se muestra un fragmento en la Tabla 5.1. También incluía una imagen PNG de los escudos de cada uno de los municipios.

**Tabla 5.1** Fragmento de los datos introducidos en la base de datos para los ejemplos de validación.

municipio	input_text
Huelva	Huelva   21.88 = PSOE   32.54   9   5 menos && PP   26.98   8   4 más && MRH   20.63   6   4 más && ADELANTE   16.67   4   2 más && Cs   3.17   0   3 menos
Quesada	Quesada   12.12 = PP   56.44   7   0 && PSOE   43.56   6   0
Teruel	Teruel   97.98 = PP   33.58   8   1 más && PAR   22.85   5   3 más && PSOE   15.23   3   2 menos && CHA   12.19   3   2 más && VOX   7.62   1   0 && PODEMOS-EQUO   7.62   1   <null> && I.U.   0.46   0   <null> && Cs   0.46   0   3 menos
(...)	(...)

Estos tres primeros son los que utilizaremos en el Caso 1, Caso 2 y Caso 3 de este capítulo.

### 5.2 Caso 1: 2 partidos de los cuales 2 estatales

Empezaremos con un caso en el que el usuario hace cuestiones sobre el municipio de Quesada. Los datos que nos hemos inventado sobre los resultados de las elecciones en esta localidad son para un escrutinio del

12.12 %:

- PP (Partido estatal) - 32.54 % de los votos - 9 concejales (mismo resultado que en 2019)
- PSOE (Partido estatal) - 26.98 % de los votos - 8 concejales (mismo resultado que en 2019)

La primera prueba comienza iniciando el subasistente indicándoselo a *Google Assistant*. Las capturas pueden verse en la Figura 5.1 y Figura 5.2. Zuri comienza con un mensaje de bienvenida indicando lo que puede hacer. En próximas capturas veremos que este mensaje va cambiando para proporcionar naturalidad y dinamismo a la aplicación. Tras ello, el usuario pide un resumen de la situación actual en el municipio y el asistente le responde de forma acertada, pues lo que esperamos de un resumen es que nos diga quién va ganando, entre otras cosas. A continuación se solicita más información, y se responde con el artículo generado completo. El texto es totalmente acertado, sin ningún tipo de error, y mostrando una tarjeta con el escudo del municipio de Quesada. Además, hay que señalar cómo el modelo ha aprendido a redactar textos periodísticos, utilizando algunos recursos como el de cambiar PP por "formación popular" para no repetir el nombre del partido.



quiero hablar con Zuri

Vale. Voy a buscarte la última versión de Zuri.

¡Hey! Me llamo Zuri, soy un asistente virtual que te ayudará a no perderte nada de los resultados de las elecciones municipales de 2023. Puedes preguntarme cosas como... ¿Cómo andan las elecciones municipales en Melilla?, o, ¿cuántos concejales obtendría EH Bildu en Urretxu?, o, ¿cuál es el porcentaje de voto escrutado en Reus?

qué tal va el reparto de concejales en Quesada

El PP triunfaría en Quesada con siete concejales, lo cual se traduciría en una cantidad similar que en los anteriores comicios. ¿Quieres saber más?

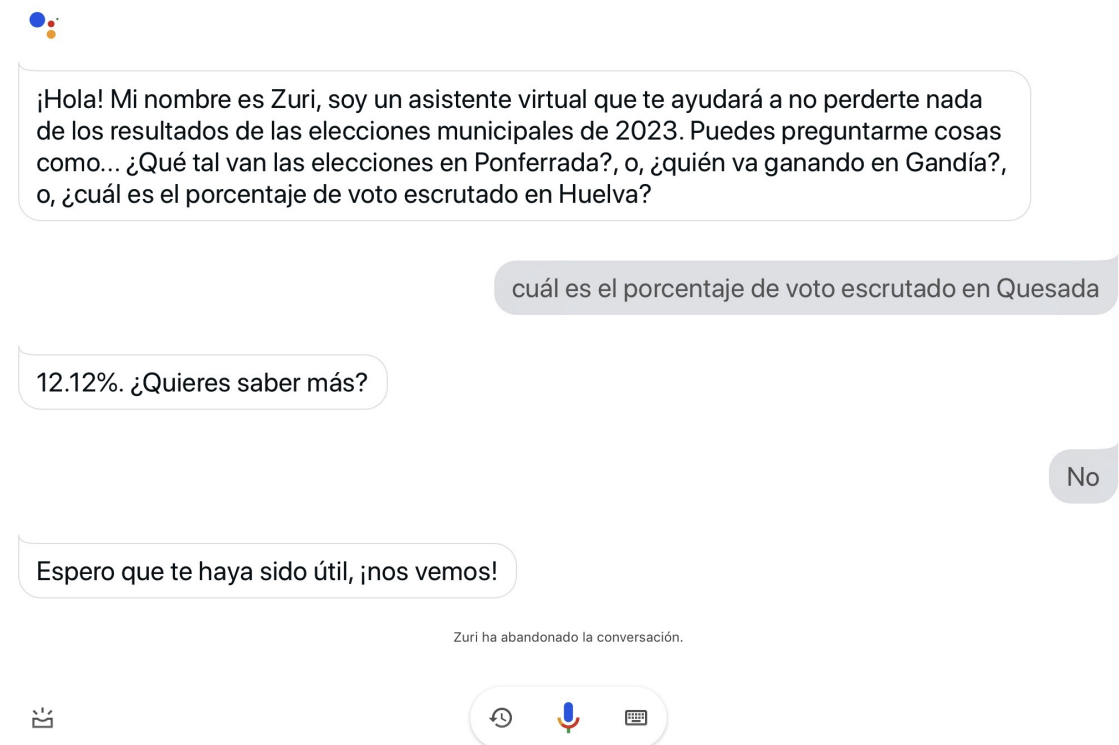


**Figura 5.1** Captura de la aplicación en funcionamiento. Caso 1. Solicitud de titular y ampliación de información. Parte 1.

La otra prueba que se hizo fue que, tras iniciar el asistente, se pregunta por el porcentaje de escrutinio actual, lo cuál puede ser útil para saber si el recuento está avanzado o no. Zuri responde a esta pregunta específica de manera exitosa y breve y, al no querer el usuario una ampliación de la información, se despide y cierra la aplicación. Destaca de forma positiva cómo el modelo sabe buscar la información, aunque la pregunta habla de porcentaje de voto escrutado y en el texto dice recuento de votos. En la Figura 5.3 se aprecia una captura de esta prueba que hemos descrito.



**Figura 5.2** Captura de la aplicación en funcionamiento. Caso 1. Solicitud de titular y ampliación de información. Parte 2.



**Figura 5.3** Captura de la aplicación en funcionamiento. Caso 1. Pregunta específica sobre el porcentaje de voto escrutado.

### 5.3 Caso 2: 5 partidos de los cuales 3 estatales, 1 regional, 1 local

En este caso, el usuario preguntará información sobre el municipio de Huelva capital. Según los datos de entrada que hemos diseñado para la prueba, se obtendrían los siguientes resultados en la ciudad para un escrutinio del 21.88 %:

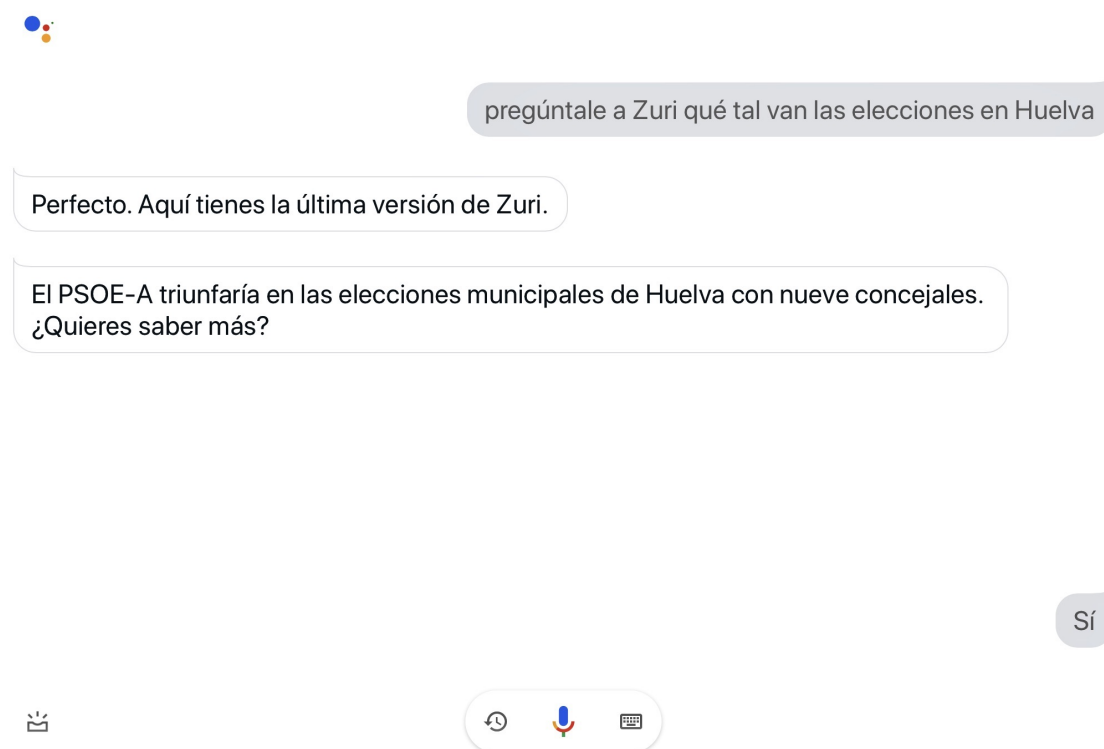
- PSOE (Partido estatal) - 32.54% de los votos - 9 concejales (5 menos que en 2019)
- PP (Partido estatal) - 26.98% de los votos - 8 concejales (4 más que en 2019)
- MRH (Partido local) - 20.63% de los votos - 6 concejales (4 más que en 2019)
- ADELANTE (Partido regional) - 16.67% de los votos - 4 concejales (2 más que en 2019)
- Cs (Partido estatal) - 3.17% de los votos - 0 concejales (3 menos que en 2019)

En la Figura 5.4 y Figura 5.5 se muestran capturas de la primera prueba. El usuario comienza haciendo directamente una petición de un resumen corto de los datos, sin iniciar previamente el subasistente de voz. Así, este muestra un titular inicial de los resultados, acertando con un resumen certero y breve de la situación actual.

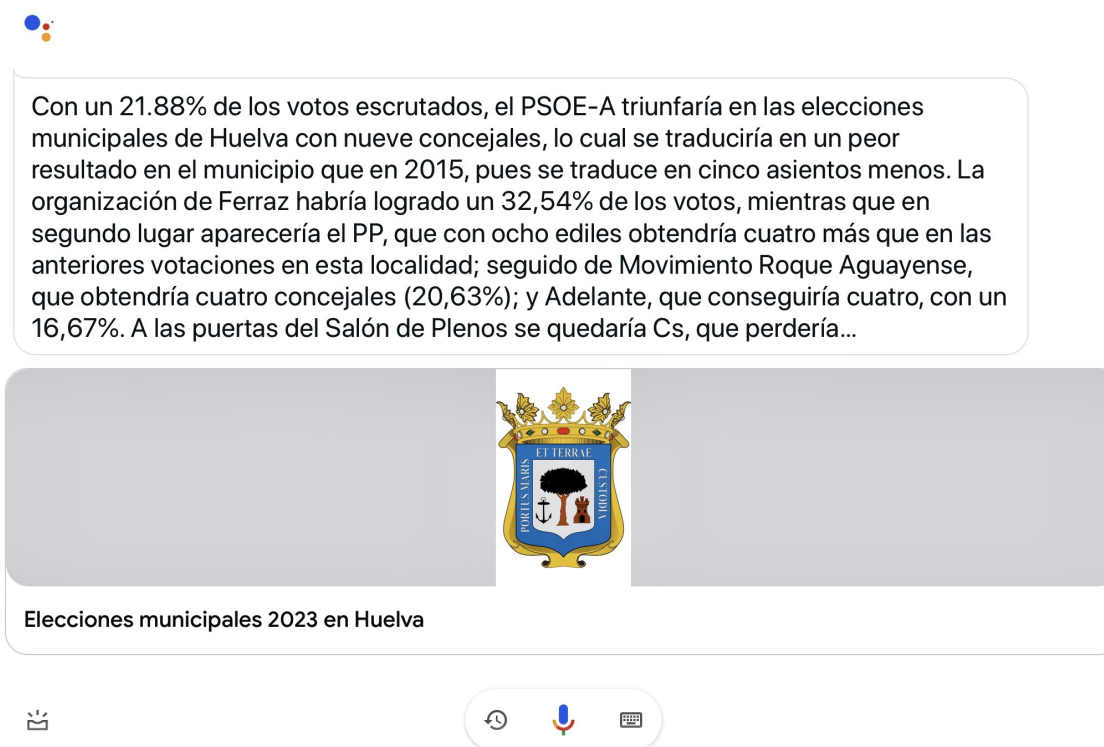
A continuación, tras solicitar el usuario más información, muestra el artículo completo junto con una tarjeta de información con el escudo de la ciudad. Aunque en la captura se ve el texto cortado, el asistente de voz lo lee entero, completándolo con: "A las puertas del Salón de Plenos se quedaría Cs, que perdería todos sus miembros del Cabildo, con un 3,17%.". Si nos fijamos en el texto, vemos que es capaz de elaborar un texto con sentido, ameno y con la gran mayoría de datos correctos. Es incluso capaz de utilizar seudónimos para no repetir el nombre de un partido, por ejemplo dice "la organización de Ferraz" refiriéndose al PSOE. Sin embargo, señalamos tres errores:

1. Se refiere a las anteriores elecciones como las de 2015. Realmente se refiere a las de 2019. Esto se debe a que fue entrenado con los datos de 2019 y 2015, y tiene esa referencia de año al hablar de las anteriores elecciones.
2. Con MRH, al ser un partido local del que no tiene tantas referencias, se equivoca al nombrar su nombre completo, confundiendo con otra formación política, el Movimiento Roque Aguayense, cuando realmente se trata de la Mesa de la Ría de Huelva.
3. Con la misma MRH, dice que obtendría 4 concejales cuando realmente obtendría 6. Quizás confunde el dato con el de que ha obtenido 4 más que en 2019.

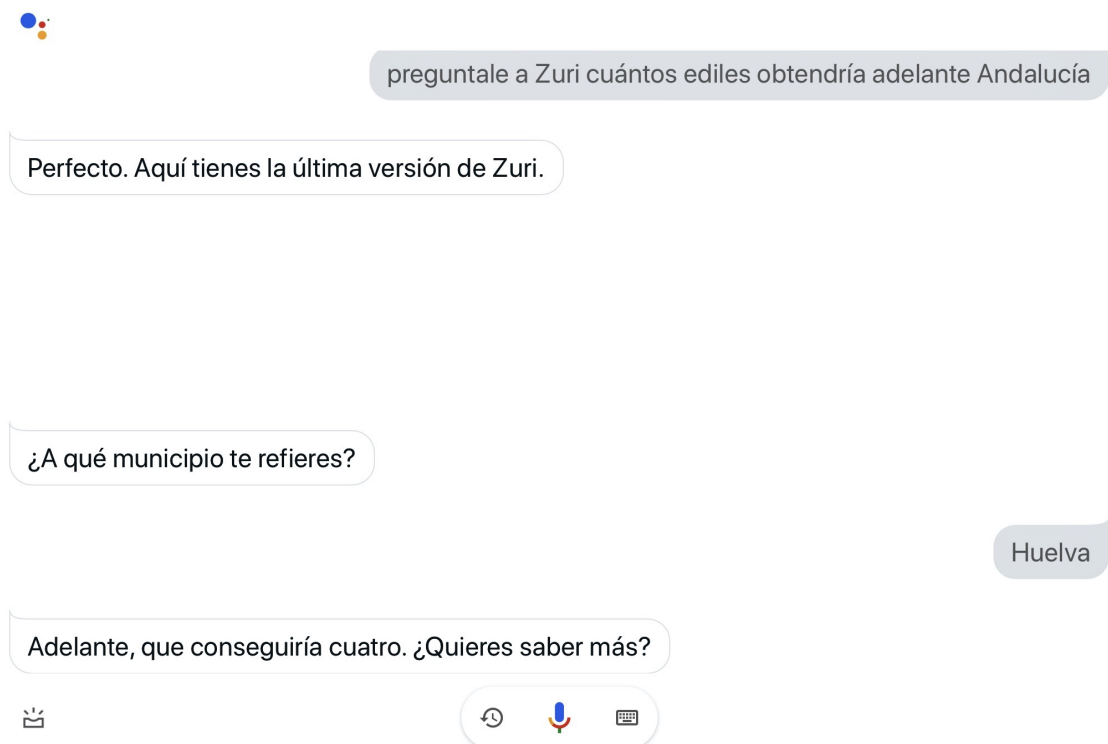
Otra prueba que se hizo con este mismo municipio fue la de empezar preguntándole algo específico, concretamente cuántos concejales obtendría el partido Adelante Andalucía con el escrutinio actual. La captura es apreciable en la Figura 5.6. Esta vez se prueba a preguntarle sin indicar el nombre de municipio en un primer momento. Como vemos, la aplicación lo detecta y simplemente le pregunta al usuario para recoger esa información que le es imprescindible. Fijándonos en la respuesta, el sistema acierta y nos da la información. Es destacable como aunque el partido figura en la base de datos únicamente como "ADELANTE" no se confunde al indicarlo como "Adelante Andalucía". Sin embargo, aunque correcto, el texto para proporcionar la información queda algo antinatural. Esto se debe a que es un trozo del artículo completo que el modelo ha cogido para elaborar la respuesta. Tras ello, nos pregunta si queremos ampliar la información y si hubiéramos respondido de forma afirmativa, habríamos obtenido lo mismo que en la Figura 5.5.



**Figura 5.4** Captura de la aplicación en funcionamiento. Caso 2. Solicitud de titular y ampliación de información. Parte 1.



**Figura 5.5** Captura de la aplicación en funcionamiento. Caso 2. Solicitud de titular y ampliación de información. Parte 2.



**Figura 5.6** Captura de la aplicación en funcionamiento. Caso 2. Pregunta específica sobre el número de escaños de un partido .

#### 5.4 Caso 3: 8 partidos de los cuales 6 estatales, 2 regionales

En este último caso, todas las interacciones con el asistente irán referidas a la ciudad de Teruel, la cual, según los datos diseñados para las pruebas, manejaría estos resultados para un escrutinio del 97.98 %:

- PP (Partido estatal) - 33.58 % de los votos - 8 concejales (1 más que en 2019)
- PAR (Partido regional) - 22.85 % de los votos - 5 concejales (3 más que en 2019)
- PSOE (Partido estatal) - 15.23 % de los votos - 3 concejales (2 menos que en 2019)
- CHA (Partido regional) - 12.19 % de los votos - 3 concejales (2 más que en 2019)
- VOX (Partido estatal) - 7.62 % de los votos - 1 concejal (mismo resultado que en 2019)
- PODEMOS-EQUO (Partido estatal) - 7.62 % de los votos - 1 concejal (no hay datos de comparación con 2019)
- I.U. (Partido estatal) - 0.46 % de los votos - 0 concejales (no hay datos de comparación con 2019)
- Cs (Partido estatal) - 0.46 % de los votos - 0 concejales (no hay datos de comparación con 2019)

La primera de las pruebas comienza, como en los casos anteriores, indicándole a Zuri la voluntad de tener un titular resumen de la información que se conoce sobre los resultados. Las capturas de esta prueba se pueden ver en la Figura 5.7 y Figura 5.8. De nuevo, a esta primera petición el asistente responde de forma correcta y directa. Tras ella, el usuario pide el texto completo. El artículo es, en su mayoría, correcto y sin errores de escritura. En este caso el segundo partido era regional y no ha dado ningún tipo de problema. No obstante, hay tres errores en los que nos podemos fijar:

1. Faltan los datos de porcentaje de votos de la segunda y tercera formación política. Esto no es un error como tal, pero si es algo a señalar ya que puede dar a un texto incompleto en información.
2. El texto indica que la fuerza regionalista CHA obtuvo un concejal, cuando realmente fueron 3. No encontramos explicación para este fallo.
3. PODEMOS-EQUO, aunque obtiene un concejal, no aparece en todo el artículo. Esto es un error importante, pues ningunea la presencia de un partido político. El alto número de partidos que aparecían



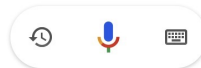
en este caso puede ser el causante, ya que quizás en los datos con los que se entrenó al modelo no había muchos ejemplos de municipios con un alto número de formaciones políticas representadas.



pregúntale a Zuri cómo andan las elecciones en Teruel

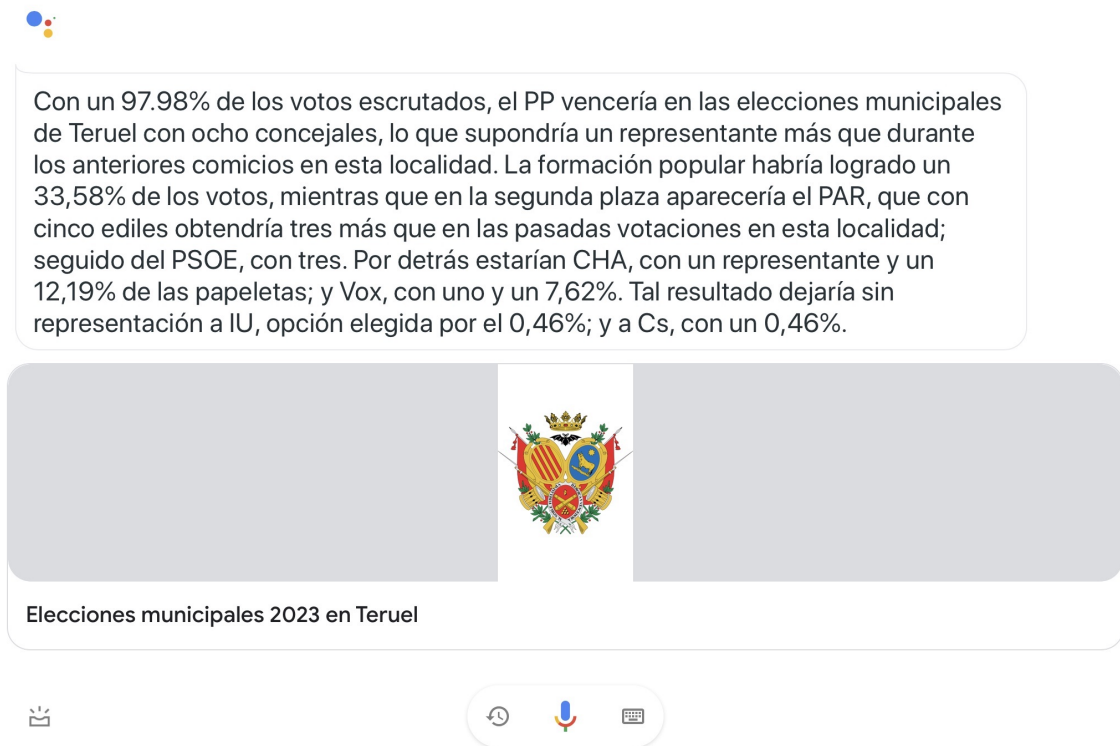
El PP vencería en las elecciones municipales de Teruel con ocho concejales. ¿Quieres saber más?

sí

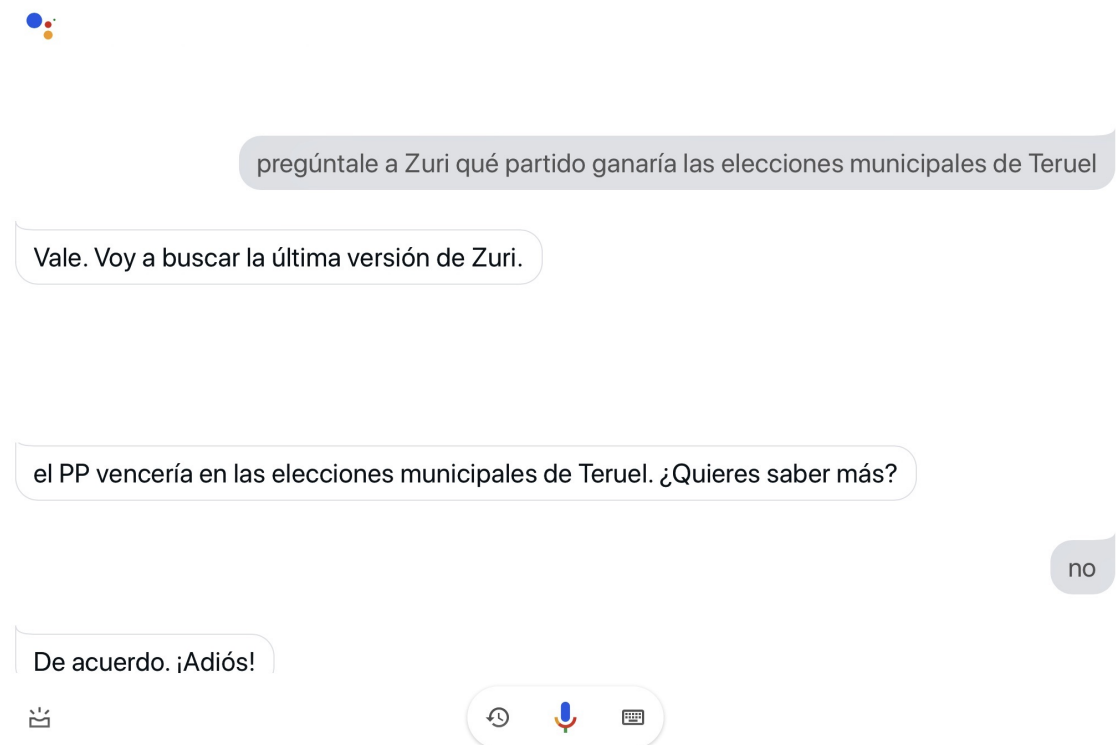


**Figura 5.7** Captura de la aplicación en funcionamiento. Caso 3. Solicitud de titular y ampliación de información. Parte 1.

La otra prueba será una pregunta específica a Zuri. En este caso, se pregunta por el partido concreto que ha ganado las elecciones (el PP en estos datos de ejemplo). Se aprecia la captura en la Figura 5.9. El modelo acierta con su respuesta, aunque con una falta de ortografía, pues inicia la oración con la primera letra en minúscula. Por lo demás, todo estaría correcto.



**Figura 5.8** Captura de la aplicación en funcionamiento. Caso 3. Solicitud de titular y ampliación de información. Parte 2.



**Figura 5.9** Captura de la aplicación en funcionamiento. Caso 3. Pregunta específica sobre partido vencedor de las elecciones .

## 6 Conclusiones y Desarrollos Futuros

---

**T**ras quedar explicados y detallados todos los aspectos que rodeaban al desarrollo de este proyecto, en este capítulo final describiremos las conclusiones a las que hemos llegado con la finalización de la aplicación y aquellas mejoras que podría tener.

En primer lugar, podemos decir que los objetivos declarados en la introducción de este trabajo han sido cumplidos con éxito. Se ha conseguido desarrollar una aplicación para la cobertura de elecciones municipales, que interactúa de manera adecuada con el usuario y que genera artículos con sentido y en la mayoría de casos con datos concretos y acertados. Además, la plataforma utilizada, un asistente de voz, ha resultado de gran utilidad. Durante el desarrollo del trabajo se han aprendido múltiples conceptos de Inteligencia Artificial que no habían sido tocados durante el grado de Ingeniería de las Tecnologías de Telecomunicación y esta memoria ha servido para explicar y divulgar el proceso de diseño e implementación.

Centrándonos en la aplicación resultante del proyecto, se pueden destacar cuatro de sus mejores características a modo de conclusión: es **actual**, pues está diseñada para asistentes de voz los cuáles se encuentran en auge y están protagonizando el futuro de la tecnología; proporciona **accesibilidad**, cualquier persona puede manejarla y está disponible en multitud de dispositivos, lo cual es beneficioso para el fin de servicio público del periodismo; garantiza **versatilidad**, ya que podría adaptarse fácilmente para otros muchos ámbitos, como los resultados de partidos de fútbol locales, el nivel de contaminación en zonas concretas de un municipio, etc.; y es totalmente **original** la fusión de tecnologías que se ha dado para su desarrollo.

No obstante, hay que mencionar que la aplicación tiene amplios márgenes de mejora, que a continuación se describen:

- No funciona de forma perfecta en todos los casos. Hay veces que da datos erróneos o simplemente los obvia. Esto, como se ha comentado, tiene su ámbito de mejora en el entrenamiento del modelo de lenguaje. Al fin y al cabo, se han usado sólo 2000 secuencias de datos para entrenarlo, por lo que si esto se amplía, con una mejor selección de los mismos y a gran escala, los resultados deberían mejorar.
- El artículo generado no deja de ser un párrafo que sólo habla de los datos concretamente. Se podría hacer un texto más largo y completo, que se asemejase más a los de un artículo periodístico escrito por personas, en los que se hiciera un desarrollo y análisis mejor de los resultados. Por ejemplo, se podría intentar analizar en el artículo los datos de participación, cómo podrían ser los futuros pactos para la elección del alcalde o alcaldesa; o cómo se comparan los datos del municipio con los de otros de su alrededor.
- El asistente de voz podría complementar los textos entregados con diagramas y figuras visuales de los datos, con aplicaciones interactivas como un "pactómetro" (ver cómo podrían sumar las fuerzas políticas y llegar a la mayoría absoluta) o enlaces a otras fuentes de información.
- Las posibilidades de personalización que nos brindan las herramientas de *Google Assistant* no dejan de ser limitadas. Aspectos como la voz del asistente o la interfaz de usuario también tendrían ciertos márgenes de mejora.



# Apéndice A

## CÓDIGOS

**Código A.1** Código Python que se utilizó para construir el dataset de entrenamiento del modelo "zuri", haciendo scraping con los artículos de El Confidencial.

```
# -*- coding: utf-8 -*-
"""
Para cada uno de nuestros municipios que tenemos en el dataset, obtendremos el
texto de entrenamiento de la página de El Confidencial, que es del tipo:

*Con un 100% de los votos escrutados, el PSOE triunfaría en las elecciones
municipales de Utebo con siete concejales, lo cual se traduciría en uno más
que los registrados en la región durante los pasados comicios. La formació
n socialista se habría hecho con el 33,2% de los votos, porcentaje que le
serviría para estar por delante del PP, que se quedaría con la segunda
plaza al obtener tres ediles; mientras que el tercer lugar correspondería a
Cs, con tres; y le seguirían IU, con dos; FIA con uno; y Vox, con uno.
Peor suerte correrían Podemos-Equo, que con un 4,75% no se sentaría en el
Salón de Plenos; y el PAR, que nuevamente se quedaría a las puertas del
concejo municipal al obtener el 4,06%.*
"""

import os
import pandas as pd
import numpy as np

file = 'train_data_asistente.csv'
data = pd.read_csv(file, index_col=0)

from googlesearch import search, get_random_user_agent
from random import randint
from bs4 import BeautifulSoup
import urllib.request
import re
import unicode
import requests
import json

def google_results(keyword):
    """ Esta función realiza la búsqueda en Google con las palabras de 'keyword'
    y devuelve el enlace del primer resultado.
```

```

Args:
    keyword (string): palabras clave a buscar

Returns:
    string: enlace al primer resultado
    """
i=1
while i<3:
    try:
        query = keyword
        query = urllib.parse.quote_plus(query) # Format into URL encoding
        google_url = "http://api.scraperapi.com?api_key=[[API-KEY]]&url=https
            ://www.google.com/search?q=" + query + "&autoparse=true"
        response = requests.get(google_url, {"User-Agent": get_random_user_agent
            })
        datosBusqueda = json.loads(response.text)
        return datosBusqueda['organic_results'][0]['link']
    except IndexError:
        i += 1
        print("IndexError.")
    except:
        print("ERROR GOOGLE DESCONOCIDO")

    """ Este bucle irá municipio por municipio buscando el enlace al artículo de El
        Confidencial, cogiendo del HTML del artículo el párrafo que nos interesa y
        guardándolo en el DataFrame."""
for indice in range(0,8121):
    print("i = " + str(indice))
    municipioString = data.at[indice, 'municipio']

    # to search
    query = "elecciones municipales 2019 " + municipioString + " site:
        elconfidencial.com/elecciones-municipales-y-autonomicas/resultados/"

    try:
        enlace = google_results(query)
        print(municipioString + ": " + enlace)

        nombre = enlace
        nombre = re.sub("https://www.elconfidencial.com/elecciones-municipales-y-
            autonomicas/resultados/2019-05-../", '', nombre)
        nombre = re.sub("-escrutinio-recuento-26m...../", '', nombre)

        municipioStringNORM = unidecode.unidecode(municipioString).lower().replace(
            " ", "-").replace("'", "-")

        if (municipioStringNORM == nombre):
            try:
                enlace = "http://api.scraperapi.com?api_key=6[[API-KEY]]&url=" +
                    enlace
                print(enlace)
                my_request = urllib.request.urlopen(enlace)
                my_HTML = my_request.read()

                try:
                    soup = BeautifulSoup(my_HTML)
                    target = soup.find(id="news-body-center").contents[2].get_text()

```

```

        print(target)
        data.at[indice, "target_text"] = target
    except:
        print("Error con el contenido")

    except:
        print("Había link pero HTTP dio error")
        data.at[indice, "target_text"] = enlace

    else:
        print("Se equivocó al buscar")

except TypeError:
    print("No resultado con " + municipioString)

except:
    print("ERROR DESCONOCIDO")

data.to_csv('train_data_prov.csv')
```

**Código A.2** Código Python utilizado para el entrenamiento del modelo "zuri".

```

""" Código con importante base del diseñado por Mathew Alexander para https://
towardsdatascience.com/data-to-text-generation-with-t5-building-a-simple-
yet-advanced-nlg-model-b5cce5a6df45 """

import pandas as pd
import os
import torch
from transformers import MT5Tokenizer, MT5ForConditionalGeneration
from transformers.optimization import Adafactor
import time
import warnings
warnings.filterwarnings('ignore')

train_df=pd.read_csv('zuri_train_data.csv')

train_df=train_df.sample(frac = 1)

batch_size=2
num_of_batches=len(train_df)/batch_size
num_of_epochs=4

num_of_batches=int(num_of_batches)

print(num_of_batches)

""" Vemos si la GPU está disponible """

if torch.cuda.is_available():
    dev = torch.device("cuda:0")
    print("Running on the GPU")
else:
    dev = torch.device("cpu")
    print("Running on the CPU")
```

```

tokenizer = MT5Tokenizer.from_pretrained("LeoCordoba/mt5-small-mlsum")
model = MT5ForConditionalGeneration.from_pretrained("LeoCordoba/mt5-small-mlsum", return_dict=True)
model.to(dev)

optimizer = Adafactor(
    model.parameters(),
    lr=1e-3,
    eps=(1e-30, 1e-3),
    clip_threshold=1.0,
    decay_rate=-0.8,
    beta1=None,
    weight_decay=0.0,
    relative_step=False,
    scale_parameter=False,
    warmup_init=False
)

num_of_epochs=4

model.train()

loss_per_10_steps=[]
for epoch in range(1,num_of_epochs+1):
    print('Running epoch: {}'.format(epoch))

    running_loss=0

    for i in range(num_of_batches):
        inputbatch=[]
        labelbatch=[]
        new_df=train_df[i*batch_size:i*batch_size+batch_size]
        for indx,row in new_df.iterrows():
            input = 'texto: ' + row['municipio'] + ' | ' + str(row['escrutinio']) + ' '
            = ' + row['input_text']+'</s>'
            labels = row['target_text']+'</s>'
            inputbatch.append(input)
            labelbatch.append(labels)
        inputbatch=tokenizer.batch_encode_plus(inputbatch,padding=True,max_length=400,return_tensors='pt')['input_ids']
        labelbatch=tokenizer.batch_encode_plus(labelbatch,padding=True,max_length=400,return_tensors="pt") ["input_ids"]
        inputbatch=inputbatch.to(dev)
        labelbatch=labelbatch.to(dev)

        # clear out the gradients of all Variables
        optimizer.zero_grad()

        # Forward propogation
        outputs = model(input_ids=inputbatch, labels=labelbatch)
        loss = outputs.loss
        loss_num=loss.item()
        logits = outputs.logits
        running_loss+=loss_num
        if i%10 ==0:
            loss_per_10_steps.append(loss_num)

```



```

    # calculating the gradients
    loss.backward()

    #updating the params
    optimizer.step()

    running_loss=running_loss/int(num_of_batches)
    print('Epoch: {} , Running loss: {}'.format(epoch,running_loss))

import matplotlib.pyplot as plt

steps = [i*100 for i in range(len(loss_per_10_steps))]

plt.plot(steps, loss_per_10_steps)
plt.title('Loss')
plt.xlabel('Steps')
plt.ylabel('Loss')
plt.show()

torch.save(model.state_dict(), path)

```

**Código A.3** Código Python que realiza el procesamiento de los datos provenientes de los resultados brutos de las elecciones para convertirlos en un formato de entrada apto para el generador.

```

# -*- coding: utf-8 -*-
"""Realizaremos el preprocesado de los datos electorales para obtenerlo en
    forma de la entrada 'input_text' que lee el generador de texto.
    """

import openpyxl
import os
import pandas as pd

"""Obtenemos la hoja de Excel, descargada de la página del Ministerio del
    Interior, con los datos de todos los municipios de las elecciones
    municipales de mayo de 2019."""

file = 'data2019.csv'
data = pd.read_csv(file)
data.info

data.head(10)

"""A continuación, calculamos el número de concejales a elegir correspondiente
    a cada municipio según la población censada, y tal como indica la Ley Orgá
    nica del Régimen Electoral General."""

concejales = 0

for i in data.index:
    municipio = data.at[i, 'municipio_nom']
    poblacion = data.at[i, 'poblacion']

    if (poblacion <= 100):
        concejales = 3
    elif (poblacion <= 250):

```

```

    concejales = 5
elif (poblacion <= 1000):
    concejales = 7
elif (poblacion <= 2000):
    concejales = 9
elif (poblacion <= 5000):
    concejales = 11
elif (poblacion <= 10000):
    concejales = 13
elif (poblacion <= 20000):
    concejales = 17
elif (poblacion <= 50000):
    concejales = 21
elif (poblacion <= 100000):
    concejales = 25
else:
    concejales = 25 + poblacion//100000
    if (concejales % 2 == 0):
        concejales += 1

# ANADIMOS LA COLUMNA EN EL DATASET
data.loc[i, 'concejales'] = concejales

print("Municipio: " + str(municipio) + " - Poblacion: " + str(poblacion))
print("  Concejales: " + str(concejales))

data.concejales = data.concejales.astype('Int64')
data.head(10)

"""Crearemos un nuevo DataFrame para guardar aquí los datos que realmente
    utilizaremos en el generador de textos. De primeras guardamos sólo el
    nombre del municipio para tener una referencia."""

data_util = data[['municipio_nom']].copy()
data_util.head(10)

"""Esta es una clase creada por Pedro Ferrer y Silvia Fuentes (https://github.
    com/vehrka/dhondt), para poder calcular los concejales (escaños) que le
    corresponden a cada partido según sus votos, aplicando la ley d'Hondt."""

import sys
from argparse import ArgumentParser

class dhondt():
    """Class to calculate d'Hondt statistics
    :Authors: Pedro Ferrer, Silvia Fuentes
    :Date: 2015-07-20
    :version: 1.1
    The minimum data to be provided is:
    + The number of seats [nseats]
    + The minimum percentage to get into the calculation [minper]
    + A dictionary with the votes of the candidatures [dcandi]
        dcandi = {'000001': 51000, '000002': 46000, '000007': 34000, '000006':
            29000, 'others': 31000}
    CAVEAT LECTOR
    + It doesn't resolve seat ties

```

```

+ Always gets rid of a party called 'others'
"""
def __init__(self, nseats, minper, dcandi, census=0, blankv=0, sploitv=0):
    self.nseats = nseats
    self.minper = minper
    self.census = census
    self.blankv = blankv
    self.sploitv = sploitv
    self.dcandi = dcandi.copy()
    self.calc()

def __repr__(self):
    candidatures = sorted(self.dcandi.items(), key=lambda p: p[1], reverse=
        True)
    return '<dhondt nseats:{0} minper:{1} candi:{2}>'.format(self.nseats,
        self.minper, candidatures)

@property
def nseats(self):
    return self.__nseats

@nseats.setter
def nseats(self, nseats):
    if type(nseats) is int and nseats > 0:
        self.__nseats = nseats
    else:
        raise AttributeError('The number or seats value must be an integer
            greater than 0')

@property
def minper(self):
    return self.__minper

@minper.setter
def minper(self, minper):
    if type(minper) is float and minper > 0:
        self.__minper = minper
    else:
        raise AttributeError('The minimum percentage value must be a float
            greater than 0')

@property
def census(self):
    return self.__census

@census.setter
def census(self, census):
    if type(census) is int:
        self.__census = census
    else:
        raise AttributeError('The census value must be an integer')

@property
def blankv(self):
    return self.__blankv

@blankv.setter

```

```

def blankv(self, blankv):
    if type(blankv) is int:
        self.__blankv = blankv
    else:
        raise AttributeError('The blank votes value must be an integer')

@property
def sploitv(self):
    return self.__sploitv

@sploitv.setter
def sploitv(self, sploitv):
    if type(sploitv) is int:
        self.__sploitv = sploitv
    else:
        raise AttributeError('The sploit votes value must be an integer')

@property
def dcandi(self):
    return self.__dcandi

@dcandi.setter
def dcandi(self, dcandi):
    if type(dcandi) is dict:
        self.__dcandi = dcandi.copy()
        try:
            sum(dcandi.values())
        except TypeError:
            raise AttributeError('The candidatures votes values must be integers')
    else:
        raise AttributeError('The candidatures data must be a dictionary')

def __mindata(self):
    if self.nseats and self.minper and self.dcandi:
        return True
    return False

def calc(self):
    """Performs the calculation"""
    if not self.__mindata():
        sys.exit('Minimum data not set')
    vtot = sum(self.dcandi.values())
    # # TODO: Finish script with the RESULTS and PARTICIPATION sections
    # ncan = len(self.dcandi)
    # if self.census < (vtot + self.blankv + self.sploitv):
    #     bvcensus = False
    #     self.census = 0
    #     nabs = 0
    # else:
    #     bvcensus = True
    #     nabs = self.census - vtot - self.blankv - self.sploitv
    # Sort the candidatures in descending number of votes
    candidature = sorted(self.dcandi.items(), key=lambda p: p[1], reverse=True)
    minvot = (((vtot + self.blankv) * self.minper) / 100) - 1
    # Filter the candidatures that have not reached the minimum

```

```

candismin = list(filter(lambda p: p[1] > minvot, candidature))
candidali = list(filter(lambda p: p[0] != 'other', candismin))
#candirest = list(filter(lambda p: p[1] < minvot + 1, candidatures))

# Prepare the lists for the calculations
candinames = [p[0] for p in candidali]
candimaxis = [p[1] for p in candidali]
canditrab = [(p[1], 0) for p in candidali]

# Prepare the dictionaries for the results
self.repre = dict(zip(candinames, [0 for name in candinames]))
self.asigna = dict(zip(candinames, [[maxi] for maxi in candimaxis]))

# Perform the seat calculation
for i in range(self.nseats):
    # Find the party with the maximum number of votes in this round
    dic01 = dict(zip(candinames, canditrab))
    odic01 = sorted(dic01.items(), key=lambda p: p[1][0], reverse=True)
    parmax = odic01[0][0]
    inparmax = candinames.index(parmax)
    maxivotos = candimaxis[inparmax]
    nseatsre = canditrab[inparmax][1]
    # This line does the magic
    canditrab[inparmax] = (maxivotos / (nseatsre + 2), nseatsre + 1)
    self.repre[parmax] = nseatsre + 1
    # Fill the asignation table dictionary
    for j, trab in enumerate(canditrab):
        self.asigna[candinames[j]].append(int(trab[0]))
    # We need to know which was the party assigned with the seat before
    # the last seat
    if i == self.nseats - 2:
        penparmax = parmax
    else:
        penparmax = parmax

# Calculate the votes needed for another seat
self.falta = {}
votult = self.asigna[parmax][-2]

for name in candinames:
    votu = self.dcandi[name]
    crep = self.repre[name]
    if name == parmax:
        # The last assigned seat gets the number differently
        crepp = self.repre[penparmax]
        votp = self.dcandi[penparmax]
        vfalta = int(votp / crepp * (crep + 1) - votu)
    else:
        cvot = self.asigna[name][-1]
        vfalta = int((votult - cvot) * (crep + 1))
    pfalta = (vfalta / votu) * 100.0
    # Stores the number of votes and the percentage over the actual
    # votes
    self.falta[name] = (vfalta, pfalta)

for i in data.index:
    row_aux = data.loc[data.index == i]

```

```

row_aux2 = row_aux.loc[:, "PSOE":data.columns[-2]]
filtro = (row_aux2 != 0).any()
row_format = row_aux2.loc[:, filtro]
row_format = row_format.sort_values(by = i, axis=1, ascending = False)
row_dict = row_format.to_dict('records')[0]

candidature = sorted(row_dict.items(), key=lambda p: p[1], reverse=True)
seats = int(row_aux.concejales.values[0])

result = dhondt(seats, 5.0, row_dict)
j = 1
for partido in row_format.columns:
    stringPartido = 'partido' + str(j)
    stringConcejales = 'concejales' + str(j)
    data_util.at[i, stringPartido] = partido

    if result.repre.get(partido, -1) != -1:
        data_util.at[i, stringConcejales] = int(result.repre.get(partido))
    else:
        data_util.at[i, stringConcejales] = 0
    j += 1
    if j == 11:
        break
    #Ponemos un maximo de 10 partidos por municipio para que no sea locura

data_util.head(10)

"""Así, hemos añadido a nuestro nuevo DataFrame los datos de los partidos y sus
concejales por municipio. Después, calculamos a partir de los votos de
cada partido y los votos válidos totales, el porcentaje que le ha
correspondido a cada uno."""

for z in range(1, 10+1):
    stringConcejales = 'concejales' + str(z)
    stringPartido = 'partido' + str(z)
    stringPorcentaje = 'porcentaje' + str(z)
    data_util[stringConcejales] = data_util[stringConcejales].astype('Int64')

    for i in data.index:
        if data_util.isnull().at[i, stringPartido] == False:
            data_util.at[i, stringPorcentaje] = data.at[i, data_util.at[i, stringPartido]
            ]/data.at[i, 'votos_validos']*100

pd.options.display.float_format = "{:,.2f}".format
data_util.head(10)

"""Ahora necesitamos calcular la diferencia de escaños que ha obtenido cada
partido en cada municipio con respecto al año anterior, en este caso las
elecciones municipales de mayo de 2015. Para ello, en primer lugar
procesaremos los datos de estas elecciones como hemos hecho con las de
2019."""

file_old = 'data2015.csv'
data_old = pd.read_csv(file_old)
data_old

for i in data_old.index:

```

```

municipio = data_old.at[i, 'municipio_nom']
poblacion = data_old.at[i, 'poblacion']

if (poblacion <= 100):
    concejales = 3
elif (poblacion <= 250):
    concejales = 5
elif (poblacion <= 1000):
    concejales = 7
elif (poblacion <= 2000):
    concejales = 9
elif (poblacion <= 5000):
    concejales = 11
elif (poblacion <= 10000):
    concejales = 13
elif (poblacion <= 20000):
    concejales = 17
elif (poblacion <= 50000):
    concejales = 21
elif (poblacion <= 100000):
    concejales = 25
else:
    concejales = 25 + poblacion//100000
    if (concejales % 2 == 0):
        concejales += 1

# ANADIMOS LA COLUMNA EN EL DATASET
data_old.loc[i, 'concejales'] = concejales

data_old.concejales = data_old.concejales.astype('Int64')

data_util_old = data_old[['municipio_nom']].copy()

data_old.head(10)

for i in data_old.index:
    row_aux = data_old.loc[data_old.index == i]
    row_aux2 = row_aux.loc[:, "PP":data_old.columns[-2]]
    filtro = (row_aux2 != 0).any()
    row_format = row_aux2.loc[:, filtro]
    row_format = row_format.sort_values(by = i, axis=1, ascending = False)

    try:
        row_dict = row_format.to_dict('records')[0]

        candidature = sorted(row_dict.items(), key=lambda p: p[1], reverse=True)
        seats = int(row_aux.concejales.values[0])

        result = dhondt(seats, 5.0, row_dict)
        j = 1
        for partido in row_format.columns:
            stringPartido = 'partido' + str(j)
            stringConcejales = 'concejales' + str(j)
            data_util_old.at[i, stringPartido] = partido

            if result.repre.get(partido, -1) != -1:
                data_util_old.at[i, stringConcejales] = int(result.repre.get(partido))

```

```

        else:
            data_util_old.at[i, stringConcejales] = 0
            j += 1

        if j == 11:
            break

    except:
        print("El municipio con indice " + str(i) + " dio error.")

for z in range(1, 10+1):
    stringConcejales = 'concejales' + str(z)
    stringPartido = 'partido' + str(z)

    data_util_old[stringConcejales] = data_util_old[stringConcejales].astype('
    Int64')
'''
for i in data_old.index:
    if data_util_old.isnull().at[i, stringPartido] == False:
        data_util_old.at[i, stringPorcentaje] = data_old.at[i, data_util_old.at[i,
            stringPartido]]/data_old.at[i, 'votos_validos']*100
    else:
        break'''
data_util_old.head(10)

data_util_old['municipio_nom'] = data_util_old['municipio_nom'].str.strip()
pd.options.display.float_format = "{:,.2f}".format
data_util_old.head(10)

"""Ahora que tenemos dos DataFrame de 2015 y 2019, los compararemos y sacaremos
    la diferencia de escaños que han obtenido los partidos."""

import re

for indice, municipio in data_util.iterrows():
    municipio_old = data_util_old.loc[data_util_old['municipio_nom'] == municipio
    ['municipio_nom']]

    if municipio_old.empty == False:
        indice_old = municipio_old.index[0]

        for z in range(1, 10+1):
            stringPartido = 'partido' + str(z)
            stringConcejales = 'concejales' + str(z)
            stringDiferencia = 'diferencia' + str(z)

            if data_util.isnull().at[indice, stringPartido] == False:

                partidoN = ""
                for column in municipio_old.columns:
                    if (data_util_old.isnull().at[indice_old, column] == False):
                        if municipio_old[column].values[0] == data_util.at[indice,
                            stringPartido]:
                            partidoN_old = column
                            cadena_N_aux = re.split('(\d+)', partidoN_old)
                            N_old = cadena_N_aux[1]

```



```

        stringConcejales_old = 'concejales' + str(N_old)

        data_util.at[indice, stringDiferencia] = data_util.at[indice,
            stringConcejales] - data_util_old.at[indice_old,
            stringConcejales_old]
    else:
        break
else:
    break

data_util.head(10)

for z in range(1, 10+1):
    stringDiferencia = 'diferencia' + str(z)
    stringPorcentaje = 'porcentaje' + str(z)
    data_util[stringDiferencia] = data_util[stringDiferencia].astype('Int64')
    data_util[stringPorcentaje] = data_util[stringPorcentaje].round(2)

data_util.to_csv('data_util.csv')
data_util

"""Ahora procedemos a transformar este dataset al formato que leerá el
    generador de textos, que deberá ser de la forma: "PSOE | 46.22 | 11 | 3 88
    PP | 16.92 | 4 | 0 88 SI SE PUEDE | 10.51 | 2 | -1 88 CCa-PNC | 10.03 | 2 |
    -3 88 Cs | 6.7 | 1 | 1 88 VECINOS POR CANDELARIA | 5.6 | 1 | 1 88
    IZQUIERDA UNIDA CANARIA | 1.74 | 0 | 0 88 NCA-AMF | 0.99 | 0 | 0"""""

import random, math
input_data_asistente = pd.DataFrame(columns=('municipio', 'escrutinio', '
    input_text'))

for indice, municipio in data_util.iterrows():
    input = ""
    escrutinio = round(random.uniform(0.99, 100), 2)

    for z in range(1, 10+1):
        stringPartido = 'partido' + str(z)
        stringPorcentaje = 'porcentaje' + str(z)
        stringConcejales = 'concejales' + str(z)
        stringDiferencia = 'diferencia' + str(z)

        if data_util.isnull().at[indice, stringPartido] == False:
            nombreInput = data_util.at[indice, stringPartido]
            porcentajeInput = data_util.at[indice, stringPorcentaje]
            concejalesInput = data_util.at[indice, stringConcejales]
            diferenciaInput = ""

            if (data_util.isnull().at[indice, stringDiferencia] == False):
                diferenciaDato = data_util.at[indice, stringDiferencia]
                if diferenciaDato > 0:
                    diferenciaInput = str(diferenciaDato) + " más"
                elif diferenciaDato < 0:
                    diferenciaInput = str(-diferenciaDato) + " menos"
                elif diferenciaDato == 0:
                    diferenciaInput = str(diferenciaDato)
            else:

```

```

        diferenciaInput = "<null>"
        partidoInput = "{nombre} | {porcentaje} | {concejales} | {diferencia} &&
            ".format(nombre=nombreInput, porcentaje=porcentajeInput, concejales=
                concejalesInput, diferencia=diferenciaInput)
        input = input + partidoInput

    input = input[0:-4]
    input_data_asistente.loc[indice] = [data_util.at[indice, 'municipio_nom'],
        escrutinio, input]

input_data_asistente

input_data_asistente.to_csv('input_data_asistente.csv')

"""Tras ello, descargamos los textos de El Confidencial, para obtener la
columna "target_text" con todos los textos. """

file = 'train_data_final.csv'
data_final = pd.read_csv(file, index_col=[0])
data_final = data_final['target_text']

data_final.head(10)

file = 'input_data_asistente.csv'
input_data_asistente = pd.read_csv(file, index_col=[0])

input_data_asistente

zuri_train_data = pd.DataFrame(columns=('municipio', 'escrutinio', 'input_text',
    'target_text'))

for indice in data_final.index:
    target_text = data_final.loc[indice]
    target_text = target_text.replace("100", str(input_data_asistente.at[indice,
        "escrutinio"]))
    zuri_train_data.loc[indice] = [input_data_asistente.at[indice, 'municipio'],
        input_data_asistente.at[indice, 'escrutinio'], input_data_asistente.at[
            indice, 'input_text'], target_text]

zuri_train_data

zuri_train_data.to_csv('zuri_train_data.csv')

```

**Código A.4** Código Python que ejecuta el servidor de aplicación para comunicarse con el asistente de voz y con los modelos de generación de texto.

```

from flask import Flask
from flask_assistant import Assistant, ask, tell, context_manager, event
import pandas as pd
import torch.nn as nn
import asyncio
import time
import requests
import threading

headers = {"Authorization": "Bearer api_OQGRQzPxYMHvPpFTuWbLUDZxNwQEaXkSFx"}

```

```

API_URL_FULL = "https://api-inference.huggingface.co/models/olirods/zuri"
API_URL_PRE = "https://api-inference.huggingface.co/models/LeoCordoba/mt5-small
    -mlsum"
API_URL_QUE = "https://api-inference.huggingface.co/models/mrm8488/distill-bert
    -base-spanish-wmm-cased-finetuned-spa-squad2-es"

app = Flask(__name__)
app.config['INTEGRATIONS'] = ['ACTIONS_ON_GOOGLE']
assist = Assistant(app, route='/', project_id="zuri-xnu9")

""" Peticiones HTTP a cada uno de los modelos """
# MODELO ZURI
def query_FULL(payload):
    response = requests.post(API_URL_FULL, headers=headers, json=payload)
    return response.json()

# MODELO SUMMARIZATION
def query_PRE(payload):
    response = requests.post(API_URL_PRE, headers=headers, json=payload)
    return response.json()

# MODELO SQUAD
def query_QUE(payload):
    response = requests.post(API_URL_QUE, headers=headers, json=payload)
    return response.json()

data = pd.read_csv('zuri_input_data.csv', index_col=[0])

async def generate(input, indice, action, question):
    """ Esta función se encarga de hacer las llamadas a los modelos de
    generación de texto en segundo plano y de guardar las respuestas en el
    DataFrame para que puedan ser obtenidas más tarde por la aplicación.

    Args:
        input (string): datos de entrada de resultados electorales para el
            generador
        indice (integer): índice correspondiente del municipio dentro del
            DataFrame
        action (integer): indica que tipo de intent se ha solicitado primero:
            0 - preview
            1 - random
        question (string): [description]
    """
    query_full = query_FULL({"inputs": str(input), "parameters" : {"max_length":
        1000}})

    gen_text = query_full[0]['generated_text']
    data.at[indice, 'full'] = gen_text

    if action == 0:
        #PREVIEW
        query_pre = query_PRE({
            "inputs": gen_text, "parameters" : {"max_length": 200}})
        print(query_pre)
        summary = query_pre[0]['summary_text']
        data.at[indice, 'preview'] = summary
    elif action == 1:

```

```

    #RANDO
    query_que = query_QUE({
        "inputs": {
            "question": question,
            "context": gen_text,
        },
    })
    answer = query_que['answer']
    data.at[indice, 'question'] = answer

    return 0

''' Gestión de la llamada a la función en segundo plano '''
def loop_in_thread(loop, input, indice, action, question):
    asyncio.set_event_loop(loop)
    loop.run_until_complete(generate(input, indice, action, question))

loop = asyncio.get_event_loop()

@assist.action('preview')
def google_preview(geocity):
    """ Esta función se ejecutará cuando el usuario le pida la intent 'preview'
        a Google
        Assistant. Realiza la petición a los modelos en segundo plano y mientras
        hace tiempo
        esperando 4 segundos y llamado a la intent auxiliar.

    Args:
        geocity (string): Parámetro de la intent. Nombre del municipio.

    Returns:
        Llamada al intent preview-results
    """
    action = 0
    question = ""

    municipio = data.loc[data['municipio'] == str(geocity)]
    indice = municipio.index[0]
    input = str(data.at[indice, 'input_text'])

    t = threading.Thread(target=loop_in_thread, args=(loop, input, indice,
        action, question))
    t.start()

    time.sleep(4)
    return event("preview-results")

@assist.action('preview-results')
def google_preview_results(geocity):
    """ Esta función se ejecutará tras la primera intent 'preview' y se
        encargará de devolver los resultados finales cuando los tenga, sino
        intentará
        seguir haciendo tiempo llamandose de nuevo a sí misma.

    Args:
        geocity (string): Parámetro de la intent. Nombre del municipio.

```

```

Returns:
    o llamada al intent random-results
    o string con la respuesta
"""
time.sleep(4)
municipio = data.loc[data['municipio'] == str(geocity)]
indice = municipio.index[0]
preview = str(data.at[indice, 'preview'])
print(preview)
context_manager.add("preview-followup")

if preview == "<null>":
    return event("preview-results")
else:
    speech_text = str(preview) + ". ¿Quieres saber más?"
    resp = ask(speech_text).suggest("Sí", "No")
    return resp

@assist.action('random')
def google_random(geocity, question):
    """ Esta función se ejecutará cuando el usuario le pida la intent 'random'
        a Google
    Assistant. Realiza la petición a los modelos en segundo plano y mientras
        hace tiempo
    esperando 4 segundos y llamado a la intent auxiliar.

    Args:
        geocity (string): Parámetro de la intent. Nombre del municipio.
        question (string): Parámetro de la intent. Pregunta que realiza el
            usuario.

    Returns:
        Llamada al intent random-results
    """
    action = 1

    municipio = data.loc[data['municipio'] == str(geocity)]
    print(municipio)
    indice = municipio.index[0]
    input = str(data.at[indice, 'input_text'])

    t = threading.Thread(target=loop_in_thread, args=(loop, input, indice,
        action, question))
    t.start()

    return event("random-results")

@assist.action('random-results')
def google_random_results(geocity):
    """ Esta función se ejecutará tras la primera intent 'random' y se
        encargará de devolver los resultados finales cuando los tenga, sino
        intentará
    seguir haciendo tiempo llamandose de nuevo a sí misma.

    Args:
        geocity (string): Parámetro de la intent. Nombre del municipio.

```

```

Returns:
    o llamada al intent random-results
    o string con la respuesta
"""
time.sleep(4)
municipio = data.loc[data['municipio'] == str(geocity)]
indice = municipio.index[0]
preview = str(data.at[indice, 'question'])
context_manager.add("random-followup")

if preview == "<null>":
    return event("random-results")
else:
    speech_text = str(preview) + ". ¿Quieres saber más?"
    resp = ask(speech_text).suggest("Sí", "No")
    return resp

@assist.action('full')
def google_full(geocity):
    """ Esta función se ejecutará cuando el usuario le pida la intent 'preview'
        a Google
    Assistant. Realiza la petición a los modelos en segundo plano y mientras
        hace tiempo
    esperando 4 segundos y llamado a la intent auxiliar.

    Args:
        geocity (string): Parámetro de la intent. Nombre del municipio.
        question (string): Parámetro de la intent. Pregunta que realiza el
            usuario.

    Returns:
        Llamada al intent preview-results
    """
    municipio = data.loc[data['municipio'] == str(geocity)]
    indice = municipio.index[0]
    speech_text = str(data.at[indice, 'full'])
    resp = tell(speech_text)
    resp.card(
        title="Elecciones municipales 2023 en " + str(geocity),
        text="",
        img_url=data.at[indice, 'img'])
    return resp

if __name__ == '__main__':
    app.run('localhost', 8000)

```

# Índice de Figuras

---

2.1	Arquitectura simplificada de un sistema de PLN	4
2.2	Esquema de la estructura en capas de una red neuronal simplificada	5
3.1	Diagrama secuencial de los componentes de un modelo de lenguaje basado en la arquitectura <i>Transformer</i> [10]	8
4.1	Captura del artículo publicado por El Confidencial sobre los resultados de las elecciones municipales de 2019 en el municipio de Puebla de Sanabria (Zamora), generado por IA [19]	13
4.2	Diagrama de componentes de la aplicación en funcionamiento, así como las distintas comunicaciones que se producen entre sí. En negrita aquellos componentes que pertenecen a los autores de este proyecto.	15
4.3	Captura del <i>intent</i> "preview" de la aplicación del subasistente virtual de Google Assistant, "Zuri", en la plataforma <i>DialogFlow</i> .	16
4.4	Diagrama de flujo del funcionamiento del subasistente de voz "Zuri" en función de las peticiones del usuario y las <i>intents</i> activadas.	24
4.5	Captura del redireccionador ngrok en ejecución en un terminal del servidor de la aplicación.	25
5.1	Captura de la aplicación en funcionamiento. Caso 1. Solicitud de titular y ampliación de información. Parte 1	28
5.2	Captura de la aplicación en funcionamiento. Caso 1. Solicitud de titular y ampliación de información. Parte 2	29
5.3	Captura de la aplicación en funcionamiento. Caso 1. Pregunta específica sobre el porcentaje de voto escrutado	29
5.4	Captura de la aplicación en funcionamiento. Caso 2. Solicitud de titular y ampliación de información. Parte 1	31
5.5	Captura de la aplicación en funcionamiento. Caso 2. Solicitud de titular y ampliación de información. Parte 2	31
5.6	Captura de la aplicación en funcionamiento. Caso 2. Pregunta específica sobre el número de escaños de un partido	32
5.7	Captura de la aplicación en funcionamiento. Caso 3. Solicitud de titular y ampliación de información. Parte 1	33
5.8	Captura de la aplicación en funcionamiento. Caso 3. Solicitud de titular y ampliación de información. Parte 2	34
5.9	Captura de la aplicación en funcionamiento. Caso 3. Pregunta específica sobre partido vencedor de las elecciones	34





# Índice de Tablas

---

3.1	Descripción de las tareas para las que está entrenado el modelo base de T5 [14]	10
4.1	Características del modelo de generación de texto propio "zuri"	19
4.2	Características del modelo de generación de texto externo "mt5-small-mlsum"	20
4.3	Características del modelo de generación de texto externo "distill-bert-base-spanish"	21
4.4	Número de concejales a repartir en un municipio según el número de habitantes en las elecciones municipales españolas [45]	22
5.1	Fragmento de los datos introducidos en la base de datos para los ejemplos de validación	27



# Bibliografía

---

- [1] S. BIRD, E. KLEIN, y E. LOPER, *Natural Language Processing with Python*. O'Reilly Media, Inc., 2009.
- [2] VARIOS. Procesamiento del Lenguaje Natural. Instituto de Ingeniería del Conocimiento - UAM. 2020. [En línea]. Disponible en: <https://www.iic.uam.es/inteligencia-artificial/procesamiento-del-lenguaje-natural/>
- [3] VARIOS. El modelo de redes neuronales. IBM. 2016. [En línea]. Disponible en: <https://www.iic.uam.es/inteligencia-artificial/procesamiento-del-lenguaje-natural/>
- [4] VARIOS. Red neuronal artificial. Wikipedia. 2017. [En línea]. Disponible en: [https://es.wikipedia.org/wiki/Red\\_neuronal\\_artificial](https://es.wikipedia.org/wiki/Red_neuronal_artificial)
- [5] VARIOS. What Is Transfer Learning? Exploring the Popular Deep Learning Approach. Built In. 2019. [En línea]. Disponible en: <https://builtin.com/data-science/transfer-learning>
- [6] VARIOS. Train Deep Learning Network to Classify New Images. Mathworks. 2018. [En línea]. Disponible en: <https://es.mathworks.com/help/deeplearning/ug/train-deep-learning-network-to-classify-new-images.html>
- [7] E. HAJIRAMEZANALI, S. Z. DADANEH, A. KARBALAYGHAREH, M. ZHOU, y X. QIAN, "Bayesian multi-domain learning for cancer subtype discovery from next-generation sequencing count data," 2018. [En línea]. Disponible en: <https://arxiv.org/abs/1810.09433>
- [8] J. VINCENT, "OpenAI has published the text-generating AI it said was too dangerous to share." The Verge, 2019. [En línea]. Disponible en: <https://www.theverge.com/2019/11/7/20953040/openai-text-generation-ai-gpt-2-full-model-release-1-5b-parameters>
- [9] A. VASWANI, N. SHAZEER, N. PARMAR, J. USZKOREIT, L. JONES, A. N. GOMEZ, L. U. KAISER, y I. POLOSUKHIN, "Attention is All you Need," in *Advances in Neural Information Processing Systems*, vol. 30. Curran Associates, Inc., 2017. [En línea]. Disponible en: <https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf>
- [10] J. ALAMMAR. The Illustrated Transformer. 2018. [En línea]. Disponible en: <https://jalammar.github.io/illustrated-transformer/>
- [11] U. ANKIT. Transformer Neural Network: Step-By-Step Breakdown of the Beast. Towards Data Science. 2020. [En línea]. Disponible en: <https://towardsdatascience.com/transformer-neural-network-step-by-step-breakdown-of-the-beast-b3e096dc857f>
- [12] A. RADFORD, J. WU, R. CHILD, D. LUAN, D. AMODEI, y I. SUTSKEVER, "Language Models are Unsupervised Multitask Learners," 2019. [En línea]. Disponible en: [https://cdn.openai.com/better-language-models/language\\_models\\_are\\_unsupervised\\_multitask\\_learners.pdf](https://cdn.openai.com/better-language-models/language_models_are_unsupervised_multitask_learners.pdf)
- [13] C. RAFFEL, N. SHAZEER, A. ROBERTS, K. LEE, S. NARANG, M. MATENA, Y. ZHOU, W. LI, y P. J. LIU, "Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer," 2019. [En línea]. Disponible en: <https://arxiv.org/abs/1910.10683>

- [14] D. HRISHABH. Google's Text-To-Text Transfer Transformer (T5). Towards Data Science. 2021. [En línea]. Disponible en: <https://towardsdatascience.com/hands-on-googles-text-to-text-transfer-transformer-t5-with-spark-nlp-6f7db75cecff>
- [15] J. DEVLIN, M.-W. CHANG, K. LEE, y K. TOUTANOVA. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. 2019. [En línea]. Disponible en: <https://arxiv.org/pdf/1810.04805.pdf>
- [16] R. HOREV, "BERT Explained: State of the art language model for NLP." Towards Data Science, 2018. [En línea]. Disponible en: <https://towardsdatascience.com/bert-explained-state-of-the-art-language-model-for-nlp-f8b21a9b6270>
- [17] VARIOS, "Generación automática de textos periodísticos," in *La inteligencia artificial aplicada a informativos*. Cátedra RTVE-UAB Innovación de los Informativos en la Sociedad Digital, 2020. [En línea]. Disponible en: [http://www.gabinetecomunicacionyeducacion.com/sites/default/files/field/adjuntos/informe\\_02\\_generaciontextos\\_3\\_baja.pdf](http://www.gabinetecomunicacionyeducacion.com/sites/default/files/field/adjuntos/informe_02_generaciontextos_3_baja.pdf)
- [18] M. CARLSON, "The Robotic Reporter," *Digital Journalism*, vol. 3, no. 3, pp. 416–431, 2015. [En línea]. Disponible en: <https://doi.org/10.1080/21670811.2014.976412>
- [19] VARIOS, "Resultados de las elecciones municipales 2019 en Puebla de Sanabria." El Confidencial, 2019. [En línea]. Disponible en: [https://www.elconfidencial.com/elecciones-municipales-y-autonomicas/resultados/2019-05-27/puebla-de-sanabria-escrutinio-recuento-26m\\_2036451/](https://www.elconfidencial.com/elecciones-municipales-y-autonomicas/resultados/2019-05-27/puebla-de-sanabria-escrutinio-recuento-26m_2036451/)
- [20] VARIOS. Siri does more than ever. Even before you ask. Apple. [En línea]. Disponible en: <https://www.apple.com/siri/>
- [21] VARIOS. Amazon Alexa Voice AI | Alexa Developer Official Site. Amazon. [En línea]. Disponible en: <https://developer.amazon.com/alexa>
- [22] VARIOS. Google Assistant, your own personal Google. Google. [En línea]. Disponible en: <https://assistant.google.com>
- [23] R. AGUILAR. Cómo acceder a contenidos exclusivos de Operación Triunfo 2020 con Google Assistant. Xataka Android. 2020. [En línea]. Disponible en: <https://www.xatakandroid.com/aplicaciones-android/como-acceder-a-contenidos-exclusivos-operacion-triunfo-2020-google-assistant>
- [24] VARIOS. ngrok - secure introspectable tunnels to localhost. [En línea]. Disponible en: <https://ngrok.com>
- [25] VARIOS. Hugging Face - The AI community building the future. [En línea]. Disponible en: <https://huggingface.co/>
- [26] L. I. CÓRDOBA, "LeoCordoba/mt5-small-mlsum." Hugging Face. [En línea]. Disponible en: <https://huggingface.co/LeoCordoba/mt5-small-mlsum>
- [27] M. ROMERO. mrm8488/distill-bert-base-spanish-wwm-cased-finetuned-spa-squad2-es. Hugging Face. [En línea]. Disponible en: <https://huggingface.co/mrm8488/distill-bert-base-spanish-wwm-cased-finetuned-spa-squad2-es>
- [28] VARIOS. DialogFlow Documentation. Google Cloud. [En línea]. Disponible en: <https://cloud.google.com/dialogflow/docs>
- [29] W. MCKINNEY. pandas - Python Data Analysis Library. [En línea]. Disponible en: <https://pandas.pydata.org>
- [30] J. D. HUNTER. Matplotlib: Python plotting - Matplotlib 3.4.3 documentation. [En línea]. Disponible en: <https://matplotlib.org/>
- [31] T. OLIPHANT. NumPy. [En línea]. Disponible en: <https://numpy.org/>
- [32] A. PAZKE, S. GROSS, S. CHINTALA, y G. CHANAN. PyTorch. [En línea]. Disponible en: <https://pytorch.org/>

- [33] T. WOLF, L. DEBUT, V. SANH, J. CHAUMOND, C. DELANGUE, A. MOI, P. CISTAC, T. RAULT, R. LOUF, M. FUNTOWICZ, J. DAVISON, S. SHLEIFER, P. V. PLATEN, C. MA, Y. JERNITE, J. PLU, C. XU, T. L. SCAO, S. GUGGER, M. DRAME, Q. LHOEST, y A. M. RUSH, “Transformers: State-of-the-Art Natural Language Processing,” in *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*. Hugging Face, 2020. [En línea]. Disponible en: <https://www.aclweb.org/anthology/2020.emnlp-demos.6>
- [34] S. BIRD, E. LOPER, y E. KLEIN. Natural Language Toolkit - NLTK 3.6.2 documentation. Crummy. [En línea]. Disponible en: <https://www.nltk.org/>
- [35] T. KUDO y J. RICHARDSON, “SentencePiece: A simple and language independent subword tokenizer and detokenizer for Neural Text Processing,” pp. 66–71, Nov. 2018. [En línea]. Disponible en: <https://aclanthology.org/D18-2012>
- [36] C. SWEENEY. Welcome to Flask-Assistant! - Flask-Assistant 0.0.9. [En línea]. Disponible en: <https://flask-assistant.readthedocs.io/en/latest/index.html>
- [37] L. RICHARDSON. Beautiful Soup 4.9.0 documentation. Crummy. [En línea]. Disponible en: <https://www.crummy.com/software/BeautifulSoup/bs4/doc/>
- [38] VARIOS. urllib - URL handling modules. [En línea]. Disponible en: <https://docs.python.org/3/library/urllib.html>
- [39] E. GAZONI y C. CLARK. openpyxl - A Python library to read/write Excel 2010 xlsx/xlsm/xltx/xltm files. [En línea]. Disponible en: <https://openpyxl.readthedocs.io/en/stable/>
- [40] P.-J. FERRER y S. FUENTES. Performs d’Hondt seat calculation. GitHub. [En línea]. Disponible en: <https://github.com/vehrka/dhondt>
- [41] M. ALEXANDER. An implementation of Data-to-Text NLG model by fine-tuning T5. Towards Data Science. 2020. [En línea]. Disponible en: <https://towardsdatascience.com/data-to-text-generation-with-t5-building-a-simple-yet-advanced-nlg-model-b5cce5a6df45>
- [42] C. GARDENT, A. SHIMORINA, S. NARAYAN, y L. PEREZ-BELTRACHINI, “The WebNLG Challenge: Generating Text from RDF Data,” 2017. [En línea]. Disponible en: <https://aclanthology.org/W17-3518.pdf>
- [43] VARIOS. ScraperApi - The Proxy API for Web Scraping. [En línea]. Disponible en: <https://www.scraperapi.com>
- [44] VARIOS, “Información Electoral.” Ministerio del Interior (Gobierno de España). [En línea]. Disponible en: <http://www.interior.gob.es/informacion-electoral>
- [45] VARIOS, “LOREG. Ley Orgánica del Régimen Electoral General,” 1985. [En línea]. Disponible en: <http://www.juntaelectoralcentral.es/cs/jec/loreg/contenido>

